ARRL AMATEUR RADIO

6TH COMPUTER NETWORKING CONFERENCE

REDONDO BEACH, CALIFORNIA

AUGUST 29, 1987

6TH COMPUTER NETWORKING CONFERENCE

Coordinators:
Harold Price, NK6K
Walter A. Linstruth, WA6JPR
Paul L. Rinaldo, W4RI

Hosts:
TRW Amateur Radio Club
Southern California Digital Communications Council

American Radio Relay League
Newington, CT USA 06111

$10 in USA

ISBN: o-87259-202-2

First Edition

FOREWORD

These papers were submitted for presentation at the Sixth ARRL Amateur Radio Computer Networking Conference in Redondo Beach, California, August 29, 1987.

Papers offered at these six conferences have grown in sophistication, in keeping with the development of amateur packet radio. Topics of high interest at this time are networking, message handling and higher transmission speeds. There is ample evidence that packet radio is in the process of moving from a purely experimental activity into an operational mode with great promise for public service.

It is gratifying to see the rapid growth of amateur packet radio in North America and elsewhere. The number of TNCs sold as of mid-1987 is in excess of 30,000 worldwide.

All papers contained in this publication were submitted in camera-ready form, are unedited and are solely the responsibility of the authors.

David Sumner, K1ZZ
Executive Vice President

Newington, Connecticut
August, 1987

CONTENTS

ESTELLE: A FORMAL DESCRIPTION TECHNIQUE FOR COMMUNICATION PROTOCOLS

Michel Barbeau, VE2BPM
3360 Marechal, app. 305
Montreal, Canada
H3T 1M9

1. Introduction

A communication protocol is a set of rules for data exchange between entities of a computer network. A communication protocol may be defined, or specified, by text written in a natural language such as english. Specifications of this type are sometimes ambiguous and imprecise, mainly because today's communication protocols are very complex. Each reader makes his own assumptions and interpretations of the unclear aspects. The corresponding implementations, or concrete realizations, of the protocol, made by different groups of people, may have incompatible behaviours under certain circumstances and therefore they cannot always work together properly. In order to specify unambiguously, clearly and concisely communication protocols, what are called, Formal Description Techniques (FDT) have been developed.

Estelle [1] is a FDT developed by ISO (International Standard Organization). It is based on an Extended Finite State Machine model (EFSM). A finite state machine is a simple abstract device which has states and transitions labelled by input and output symbols. A state transition table is one of the possible ways to represent textually a FSM. FSM's are frequently used to model the control flow of systems. Communication softwares, such as data link protocols, have usually a component that can be represented by FSM's. As an example see the state transition tables of the AX.25 link-layer protocol in appendix D of [2]. Protocols have also data flow aspects involving interaction parameters, different kinds of variables and data operations. The data flow aspects are hard to represent using only FSM's. Estelle extend the idea of FSM in a sense that variables, actions and predicates, operating on those variables, are added to this basic model. The syntax of Estelle has been defined from the syntax of the programming language Pascal. New elements have been added in order to make easier the definition of aspects particular to communication protocols. In Estelle can be expressed both the control flow aspect, using FSM's, and the data flow aspect, using Pascal's elements, of a communication protocol. Protocol specifications written in the formal language Estelle are said formal with respect to informal specifications written in a natural language such as English.

Section two introduces the alternating bit, a simple data transfer communication protocol. Then, in section three, the alternating bit is used to present the FDT Estelle for the specification of communication protocols.

2. The Alternating Bit Protocol

The alternating bit is a very simple data transfer protocol which can be used, as **the** AX.243 packet-radio protocol [2], in the data link layer of the ISO reference model. This section gives a short introduction to the alternating bit.

The protocol comprises a mechanism giving the capability to recover data losses during the data transfer. It prevents also the transmitter from overloading the receiver with data. Two kinds of data blocks, or Protocol Data Units (PDU's), are exchanged between entities that are using the rules of the alternating bit. The first kind of PDU is named DT. A DT block is composed of two fields: a sequence number field and a user's data field. The second kind of PDU is named AK. AK blocks are used to acknowledge the successful transfer of DT blocks from the transmitter to the receiver. An AK PDU contains the sequence number of an acknowledged DT PDU. Values of sequence numbers alternate between zero and one. The interaction with the users of the protocol is represented by the messages SEND_request, RECEIVE_request and RECEIVE_response. The interaction RECEIVE_request has no parameter but the messages SEND_request and RECEIVE_response have both a parameter for user's data.

Figure 1 shows a typical dialogue between two entities that are using the alternating bit. The transmitter initiates the dialogue by a transmission request (1). The protocol generates a DT, numbered zero. This DT is correctly received by the peer entity (2). The user's request for data is satisfied by a RECEIVE_response which conveys the value took from the data field of the DT PDU. An acknowledgement is also sent to the transmitter. Only one DT PDU may be transmitted at once. Its successful reception must be acknowledged before transmitting more data.

Losses of PDU's are recovered. **A** second interaction SEND-request (3) generates a DT PDU numbered one. Its corresponding acknowledgement is lost during the transfer (4). After a timeout period the emitter retransmits the same DT block (5). Its reception is now

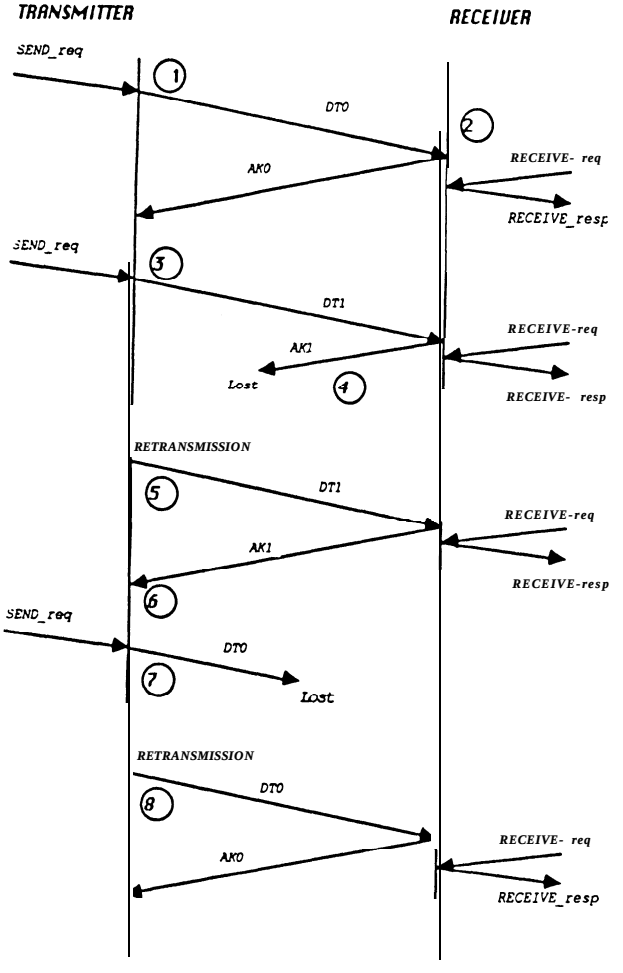


Fig. 1 Data transfer with the alternating bit

acknowledged (6). In the case where the DT block itself is lost **(7)** retransmission also occurs after a timeout period (8).

3. The Formal Specification of a Protocol

The language Estelle is used to define formally the behaviour to which, implementations of a communication protocol, might conform. However, a certain level of abstraction is kept with respect to the specific characteristics of some particular implementation. A protocol description in Estelle is generally composed of two main parts:

- The first part contains descriptions of channel types through which the protocol will exchange messages **with.** its environment (i.e. the users and the communication services provided by the layer beneath). Channels are defined in terms of the messages that can be sent by each entity communicating through it. Messages can have parameters, therefore they are defined along with their parameter identifiers and their respective data types.

- The protocol description itself is structured into one or several cooperating modules, defined in the second part of the specification. Each module corresponds to an EFSM (i.e. a FSM capable of having memory). A module may contain data type, variable and procedure declarations. Each module has interaction points, each one associated to a channel type. Communications with other modules and the protocol's environment take place at those interaction points. The control and data flow aspect of a module are essentially described by a group of transitions. **A** complete module specification consists of a module header, where its interaction points are defined, and generally **qne** module body, where data flow and control aspects are defined.

The syntax of the Estelle language is essentially an extension of the syntax of the Pascal language. New syntactic elements have been added to provide the facilities to define aspects which are specific to communication protocols. Appendix A shows the formal specification in Estelle of the alternating **bit protocol, presented in section** 2. The following is a discussion of this specification.

3.1 Channels **and Interactions**

At the beginning of the specification is declared the Pascal record *Ndata_type* describing the structure of the PDU kinds exchanged by the protocol implementations. The alternating bit uses two kinds of PDU. The fields *Id* identifies the PDU's. For a DT PDU the fields *Data* snd *Seq* are used. For an AK the field *Seq* only is used. This definition does not take into account the encoding of the PDU's in terms of bits and bytes. This information would be given in an informal specification (i.e. using a natural language). Therefore, formal and informal techniques complete each other.

Channel statements introduce the types of inter= action exchanged by the involved entities. Communication between the user of the alternating bit and the protocol will take place **over** a channel of type *U_access_point* and over a channel of type *N_access_point* between the protocol and the layer beneath (i.e. the physical layer). Over a channel of **type** *U_access_point*

the user can send the interactions *SEND_request* and *RECEIVE_request*. The protocol may, over the same channel, send the message *RECEIVE_response*. Names and data types of interaction parameters are specified in parentheses following each message identifier. Interactions such as *SEND_request*, *RECEIVE_request* and *RECEIVE_response* are generally called service primitives. The channel *N_access_point* can be interpreted in a similar fashion. Those channel declarations do not suggest any concrete realization. Services primitives may be implemented as procedure calls, subroutines linkages or interprocess communication.

3.2 Modules

Modules are defined using the *module* and *body* statements. A *module* statement is used to define what is called the module header. The header names the points of interaction with the module's environment, the channel type used at each of these points and the role attributed to the module. The module header *Alternating_bit_type* has two interaction points named *U* and *N*. The roles of the module are named: *provider* with respect to *U* and *user* with respect to N.

To each header type, are associated one or several module bodies. This way, to the same external structure may correspond different internal behaviours. The definition of each of these bodies begins with the keyword *body*. The specification of appendix A is structured into a single module. To the module header *Alternating_bit_type* corresponds a single module body identified *Alternating_bit_body*. Generally specifications of complex protocols are structured into many modules and those modules may themselves embed submodule declarations.

The skeleton of a module body is a finite state machine. Because it is difficult and generally impossible to specify a fairly complex protocol using only FSM's, elements of the Pascal language have been added to extend this basic concept. Usually a module body comprises three consecutive major sections: a declaration part, an initialization part and a transition part.

Any kind of declaration that could be found in Pascal programs may be used in an Estelle specification. The body *Alternating_bit_body* contains declarations of constants, data types, variables, procedures and functions.

The representation of the data type *Buffer_type* is undefined. Instead the definition has been replaced by the symbol "...". The body of some procedures and functions is defined as *external* or *primitive*. Those undefined aspects of the specification are left to the protocol implementers. They will choose themselves the more suitable representation and the algorithms to manage data buffers.

The state of a module, during its execution, is characterized by the values of its variables. One of these, the variable state, plays a key function. It will save the major state name (i.e. either ACK_WAIT or ESTAB in this case). This variable corresponds to the FSM component of the module. The other variables save sequence numbers and data elements, they are called context variables.

The initial state of a module is defined in the initialization part. First, this part specifies the initial major state name using the statement, *to*. Then, assignment statements give initial values to the context variables.

3.3 Transitions

The transitions define the potential state changes of a module. The module *Alternating_bit_body* has five transition types. The first three are related to data transmission. The last two handle the reception of data. Each transition has a *begin-end* bloc, similar to a Pascal procedure. This bloc is preceded by a sequence of clauses specific to Estelle. The clauses *from* and *to* express the major state change made when the transition is executed. The firing of the transition depends of the current major state and also on the truth value of a predicate specified in the *provided* clause. This predicate contains references to context variables. The firing may also depend on the arrival of an input interaction which is selected by the clause *when*, otherwise the transition is called spontaneous.

In brief, a module makes a transition if one of its transitions is enabled. A transition is enabled if the conditions jointly specified by the clauses *from*, *when* and *provided* are satisfied. The transition thru a state, first specified by the clause *to* then by the new values assigned to the context variables in the *begin-end* bloc, will be made. Any Pascal statement may be used in the *begin- end* bloc. Moreover, output messages can be generated using *output* statements.

A message sent by an output interaction is put in a queue associated to its receiver. The receiver will retrieve the message from the queue when he will be able to process it. With the statement *stateset* may be declared sets of major state names. References to these state sets can be made in the *from* clauses.

Estelle contains also statements for dynamic management of modules instances (i.e. creation, destruction, etc.). Moreover, Estelle comprises to whole standard

Pascal language. For a complete coverage of Estelle see [1].

4. Conclusion

FDT's are the first step toward the development of well defined and reliable communication softwares. FDT's provide clear and precise definition of communication protocols to implementers. Estelle is a FDT developed by IS0 that is expected to be extensively used in the future.

References

[1] ISO/TC 97/SC 21, *Estelle - A Formal Description Technique Based on an Extended State Transition Model*, DP 9074, 1986.

[2] FOX L. Terry, *AX.25 Amateur Packet-Radio Link-Layer Protocol*, ARRL, 1984.

Appendix **A: An Example Protocol Specification**

This specification has been extracted from reference [1]. Comments are placed, as in Pascal, in curly brackets. Note that this protocol definition may contain deadlocks.

```
specification Altbit;

type
    U-Data-type = . . . . { user data }
    Seq-type = 0..1 ; { sequence number range }
    Id-type = (DT, AK);
    Ndata_type = record
        Id : Id-type; { type of message }
        Data : U-Data-type; { user data }
        Sey : Seq_type; { sequence number }
    end;

{ Channel definitions }
channel U_access_point( User,Provider);
    by User:
        SEND-request (Udata : U-Data-type);
        REC'EIVE_request;
    by Provider:
        REC'EIVE_response( Udata : U-Data-type);

channel N_access_point(User,Provider);
    by User:
        DATA_request( Ndata : Ndata_type);
    by Provider:
        DATA_response( Ndata : Ndata-type);

{ Module header definition }
module Alternating-bit-type process;
    ip { interaction point list }
        U : U_access_point( Provider) common queue;
        N : N_access_point( User) individual queue
    end;

{ Module body definition }
body Alternating_bit_body for Alternating-bit-type;

const
    Retran_time = ...;
    { Retransmission time in seconds
    (defined by implementer) }

type
    Msg_type = record
        Msgdata : U-Data-type;
        Msgseq : Seq_type
    end;

    Buffer-type = . . . .

var
    Send-buffer, Recv_buffer : Buffer-type;
    Sendseq, Recvseq : Seq-type;
    P, Q : Msg-type;
    B : Ndata-type;

    state ACK-WAIT, ESTAB;

stateset
    EITHER = [ACK-WAIT, ESTAB];

procedure Copy(
    var To_Data:U_Data_type; From_data:U_Data_type);
external;
{ procedure provided by implementer:
copy a user data variable }

procedure Empty( var Data:U_Data_type);
primitive;
{ procedure provided by implementer:
initialize a variable holding user data
to the value no user data }

procedure Formatdata( Msg:Msg_type; var B:Ndata_type);
begin
    B.Id := DT;
    copy( B.Data, Msg.Msgdata);
    B.Seq := Msg.Msgseq;
end;
```

```
procedure Format_ack( Msg:Msg_type; var B:Ndata_type);
begin
    B.Id := AK;
    B.Seq := Msg.Msgseq;
end;

procedure Empty_buf( var Buf:Buffer_type);
primitive;
{ procedure provided by implementer:
set a buffer to empty }

procedure Store( var Buf:Buffer_type; Msg:Msg_type);
primitive;
{ procedure provided by implementer:
store a message into a buffer_type variable such that the
messages can be retrieved or removed in a FIFO manner }

procedure Remove( var Buf:Buffer_type);
primitive;
{ procedure provided by implementer:
remove the first message }

function Retrieve( Buf:Buffer_type):Msg_type;
primitive;
{ function provided by implementer:
retrieve the first message and return it;
the message is not removed }

function buffer_empty(Buf:Buffer_type):boolean;
primitive;
{ function provided by implementer:
check if a buffer contains a message }

procedure Inc_send_seq;
begin
    Send_seq := (Send_seq + 1) mod 2
end;

procedure Inc_recv_seq;
begin
    Recvseq := (Recvseq + 1) mod 2
end;

{ initialization-part }
initialize
to ESTAB { initialize major state variable to ESTAB }
begin
    { initialize context variables }
    Send_seq := 0;
    Recv_seq := 0;
    Empty_buf(Send_buffer);
    Empty_buf(Recv_buffer);
end;

{ Transition part }
{ Sending data }
trans
from ESTAB to ACK_WAIT
when U.Send_request
begin
    copy(P.Msgdata,Udata);
    P.Msgseq := Send_seq;
    Store(Send_buffer,P);
    Format_data( P,B);
    output N.DATA_request(B);
end;

trans
from ACK_WAIT to ACK_WAIT
delay (Retran_time) { timeout }
begin
    P := Retrieve( Send_buffer);
    Format_data( P,B);
    output N.DATA_request( B);
end;

trans
from ACK_WAIT to ESTA:B
when N.DATA_response
provided (Ndata.Id = AK) and (Ndata.Seq = Send_seq)
begin
    { remove acknowledged message }
    Remove(Send_buffer);
    Inc_send_seq
end;

{ Receiving data }
trans
from EITHER to same
when N.DATA_response
provided Ndata.Id = DT
begin
    copy(Q.Msgdata, Ndata.Data);
    Q.Msgseq := Ndata.Seq;
    Format_ack(Q,B);
    output N.DATA_request(B);
    if Ndata.Seq = Recv_seq then
    begin
        Store(Recv_buffer,Q);
        Inc_recv_seq
    end
end;
```

```
trans
from EITHER to same
when U.RECEIVE_request
provided not buffer-empty( Recv_buffer)
begin
    { retrieve received message }
    Q := Retrieve( Recv_buffer);
    output U.RECEIVE_response( Q.Msgdata);
    { remove message from receiving buffer }
    Remove(Recv_buffer)
end;
end; { of module body }
end. { of specification }
```

OSI: A PLAN COMES TOGETHER

J. Gordon Beattie, Jr., N2DSY
Thomas A. Moulton, W2VY

The Radio Amateur Telecommunications Society
206 North Vivyen Street
Bergenfield, NJ 07621
201-387-8896

ABSTRACT

This paper will provide an overview of the current state of services available on the Amateur Packet Radio and then offer solutions to the problems and limitations found. This will include an outline of a communications architecture for distributed computer systems using the Open Systems Interconnection Reference Model (OSI-RM). We will also provide a description of systems required to support the data transport and application needs of the Amateur Packet Network user. These OSI-based systems have been designed to inter-operate with each other and with other systems not part of the Amateur Packet Network. This-architecturally consistent approach addresses the operational objectives of Service, Managibility, and Performance. It has the further advantage of world-wide acceptance.

CURRENT STATE

The network support requirements of the "AVERAGE" Radio Amateur are nearly impossible to scope out. However, there are a few activities which are found to be of interest. These are:

Keyboard-to-Keyboard

Keyboard-to-Computer

Computer-to-Computer

Roundtable/Net

Bulletin Broadcast

DX'ing

Let us now examine these specific areas with an eye toward improvement.

Keyboard-to-Keyboard

Keyboard-to-Keyboard QSOs are difficult to obtain because of the lack of a reasonable calling mechanism. The "HEARD" list of many Switches and BBSs provide this function, but without specific knowledge of the status of a particular station.

What is needed is a network server which provides a gathering point for stations available for such QSOs.

Keyboard-to-Computer

The Keyboard-to-Computer activity is probably the best developed. Current message/file serverssuch as the Packet Radio MailBox System (PRMBS by KA2BQE) and the MBL Bulletin Board System (by WA7MBL) have added new user modes and services. These services include remote directory, file, and user log requests. Despite these advances, there are additional improvements to be made. These systems have not addressed the requirement for BINARY and error-free, block-oriented (FULL PACKET) data transfers. Some work has been done by KA2BQE and N2DSY of RATS to deal with the these requirements, but the solutions are only of a stop-gap nature. KA2BQE has written the BTOA and ATOB binary-to-ASCII and ASCII-to-binary programmes. These allow the contents of a binary file to be converted to and from ASCII for transmission in a standard BBS message envelope. KA2BQE and N2DSY have also worked on the parameter setting for BBS TNCs. These TNCs send sequences of FULL PACKETS instead of a packet per line. The transmission of packets is based on full packet buffers or a timeout. This approach works fairly well except when another user attempts connection to the already busy BBS. This causes a TNC message "*** connect request from: W2XYZ" to be inserted into the data stream. This has been resolved through the use of a "silent" TNC-2 EPROM written by N2WX. This is a fully functional version fo the TNC-2 software that has ALL messages removed. The BBS obtains the callsign of the station by entering command mode and using the "c" command. Before this was available these text messages had to be manually removed with a text editor. Another enhancement would be the support of multiple simultaneous users. This has been added to the PRMBS package by NN2Z, but it still only allows for two users on separate ports.

What is needed is a message/file server which can support binary files, full packets and multiple users.

Computer-to-Computer

This mode of operation started as a BBS-to-BBS function, but in the last year this activity has included a rapidly growing number of users. This growth has been fueled by the large numbers of Asian MS-DOS machines available on the U.S. market. Many Amateurs are receiving electronic mail that is directly forwarded to their systems by their "HOME" BBS. This helps reduce user demand on the BBSs during prime evening hours. Another activity is file transfer between users. These transfers are often interrupted by other users of a congested channel. The result is a restart from the beginning of the file.

The file transmission restarts greatly affect the efficiency of our BBS-to-BBS forwarding and local area networks. It also Wastes the user's time.

What is needed is a robust, automatic system for the unattended transmission of files. This system must have the capability of resuming interrupted forwarding sessions and it must be common to both users and servers.

Roundtable/Net

The roundtable or net operation is a venerable activity of Amateur Radio and it is by far the most poorly supported by packet radio. Some groups operate in "unconnected" mode and use the LAN Digipeater to relay packets to all members of the group. For reliablilty each user monitors the channel to see if the digipeater retransmitted the packet. If the packet is not relayed, the user retransmits (and therefore re-types) the message. This is far from reliable. Another approach was tried by N2WX in his Gator Switch. Each user connects to the switch, then "joins" the roundtable by issuing an appropriate command. The switch copies received data packets to each link corresponding to the members of the roundtable. Thls was thoroughly tested in GATORNET and the RATS Digiplex System and was found to be too extreme. The channel would bog down when acknowldgements would collide as they were sent by each station. This approach did however ensure a reliable, if not slow, method of group communication.

What is needed here is a reliable mechanism for getting packets to the LAN server and a method for reliably getting packets to ALL members of the roundtable.

Bulletin Broadcast

This is another venerable activity which is not supported by packet radio. Many groups have examined ways of acheiving multi-point dissemination of information. The same basic mechanism is quite handy for alerting Amateurs to band openings and disasters. Some folks have taken to using the "LCALLS" and "BUDLIST" parameters for this purpose, but It causes problems for hormal activities. Many of the reliability issues of this mode of operation are common to the Roundtable ahd Net operation mentioned above.

What is needed here is a server and a method for signalling stations and reliably transferlng data to the destination stations.

DX'ing

This is the grand-daddy, the true motherhood and apple pie aspect of Amateur Radio (second only to CW, hi ! hi !) and you guessed It, packet radio doesn't support it particularly well. When we dropped the Vancouver protocol (V-l) and moved to AX.25 Level 2, packet operators here ln New Jersey were able to connect to systems in Canada and In Maryland. Then came ALL THOSE USERS ! It was spoiled...gone forever... Well that's not quite true...We now have wormholes and Fuji (or is it OSCAR ?) and other bridges, but they are pleasant patches in the fabric of the network. NET/ROM and AX.25 Level 3 (level 2.5 ?) nodes are all over the place, but NET/ROM alters callsigns and SSIDs; and AX.25 Level 3 switches issue connect strings not recognized by the new versions of PRMBS and MBL BBS software. These nodes also provide newfound connectivity (YEA !) and enormous network loading (BOO !) Somewhere out there, we have some IP users who want the responsibility for error recovery to the the private reserve of each end system and not shared with the network. We now have a mad scramble by everyone concerned to build new software functions or gateways between all these disjointed systems.

What is needed is a uniform set of communications procedures based on common, internationally recognized protocols.

COSI ARCHITECTURE

The RATS-COSI (Connected Open Systems Interconnection} approach is based on the internationally recognized communications protocols recommended by the CCITT [International Telegraph and Telephone Consultative Committee) and the IS0 (International Organization for Standardization). These bodies provided much of the original source material that went into the AX.25 Amateur Packet,-Radio Link-Layer Protocol. When the members of AMRAD and RATS met to hammer out the AX.25 Protocol they had the foresight to choose a protocol suite which would evolve in a controlled manner and provide the basis for a complete protocol set for the users of the Amateur Packet Network. Inter-operability with other Amateur and commercial networks and international acceptance were major factors in the selection of the CCITT X.25 (or IS0 8208) protocols. This has proven to be a daring and rewarding choice. It has not been without its difficulties.

Since the group first met, the CCITT and IS0 have moved rapidly to align their documents and refine the protocols in the OSI Reference Model. Inconsistencies and gaps have been filled. The most important work has been in the definition of the upper layer protocols. This now provides a consistent structure and method for applications to communicate with corresponding systems.

The COSI Architecture defines four basic LAN servers:

- Message/File
- Network Access
- Roundtable
- Broadcast

The various server functions may be combined into a single unit, but some thought should be given to channel loading when these functions are provided.

The RATS Digiplex system is in the process of evolving to the COSI Architecture. The protocols required to support this 7 layer model have matured to the point where systems based on all layer functionalities may now be developed. Companion papers provide detailed protocol specifications for the Session, Presentation, and Application layers. These will complement previous work that defined the protocols for the

Physical, Link, Network and Transport Layers.

The communications protocols used in the RATS-COSI communications architecture are outlined in the Figure "COSI Communications Architecture". These protocols provide a common method for systems to communicate.

The level 1 [Physical Layer] protocol will vary according to the speed and band used but for the purposes of clarity most users will use the Bell 202 modem to audio frequency shift an NBFM carrier (VHF/UHF).

The level 2 (Link Layer) protocol used is a the AX.25 Link-Layer Protocol. Details may be found in the ARRL publication: "AX-25 Amateur Packet-Radio Link-Layer Protocol, Version 2". Further information may be found in the COSI Implementer's Guide.

The level 3 (Network Layer) protocol is the AX.25 Packet-Layer Protocol a conforming implementation of CCITT X.25/ISO 8208. For more information see the COSI Implementer's Guide: AX.25 PLP Network Protocol Specification for the Amateur Packet Network". This layer provides a path through a network of switching nodes. It multiplexes each user's data onto a Virtual Circuit.

The level 4 (Transport Layer) protocol is the AX.224, Class 1 Protocol a conforming subset of CCITT X.224, 1984. See the COSI Implementer's Guide: "Proposal: Recommendation AX.224 Transport Protocol Specification for the Amateur Packet Network". This layer is responsible for providing reliable data streams between corresponding systems operating in an Open System environment.

The level 5 (Session Layer) protocol is AX.225 a conforming subset of CCITT X.225, 1984. See the COSI Implementer's Guide: "Proposal: Recommendation AX.225 Session Protocol Specification for the Amateur Radio Network". This protocol provides the capability to control the connection, interruption and resumption of connections used by corresponding application programmes in an Open System environment.

The level 6 (Presentation Layer) protocol is AX.226 a conforming subset of CCITT X.226/ISO 8823. Specific Presentation Contexts have been defined for systems operating in the Amateur Packet Network. These are outlined in the COSI Implementer's Guide. See: "Proposal: Recommendation AX.226 Presentation Protocol Specification for the Amateur Radio Network".

The level 7 (Application Layer) protocols are AX.ros and the RATS Transaction Protocol (RTP). AX.ros is based on the CCITT X.ros/ISO 9272/2. This protocol provides applications with a method for communicating the invocation of remote operations on correspondent Open Systems. This Application Service Element provides the framework for the operations defined in RTP. RTP is a basic set of operational primitives that can be used to manipulate data objects in an Open System. AX.ros and RTP are described in a section of the COSI Implementer's Guide entitle&: *"Application Service Elements for Open Systems Operatang in the Amateur Packet Network".

The protocols defined for use in the Presentation and Application Layers are based on the CCITT/ISO ASN.l encodings.

COSI SYSTEMS

PRMBS and PMBx

These "C" language based message/file server (PRMBS) and the computer-based user packages (PMBx) share a common software base including the user and communications interfaces. PRMBS is a "public" or shared server package. :PMBx supports one master instead of a community of users and is simpler to configure and manage than a normal BBS. PMBx greatly reduces prime time user logins to the "public" message/file server. The PMBx package also allows users to exchange messages and files directly.

The current RATS Digiplex Message/File server is -the MS-DOS based PRMBS package written by KA2BQE. Many new features are already in released versions. The rest will be added during the fall o-f 1987. These will include:

Multiple simultaneous users

Binary transfer

Priority message handling

NTS support

Multiple callsigns for interservice gateways

Distribution lists

Distributed conferencing

Interruption/resumption of data transfer

Machine-to-Machine Protocol (COSI-base&, with ROSE and RTP)

COSI-Switch

RATS has been experimenting with several switches written by N2WX. The latest is a "C" language based system. After having examined these switches in our network several changes have evolved. The most significant is a new support method for Level 2 users accessing the Level 3 switch. This approach allows the Level 2 user to type:

"C DestinationCallsign V SwitchCallsign, DestinationNodeAddress"

or

"C KA2BQE V N2DSY-3, 609010"

This approach allows the RATS Digiplex System to offer implicit addressing. This scheme has exte'nsions to allow for access/egress digipeaters, and international connections. The COSI-Switch also has the advantage of not modifying the SSID of the orignating or the destination stations. Further the multiplexing scheme is based on the CCITT X.25 and IS0 8208 Packet Level Protocols.

COSI-Net

This system offers local services to the LAN. These services include management of roundtable or net activities. The user accesses this server and then selects the net

function. Each packet from the user will be acknowledged. Then the server will transmit the data in a UI frame that has a source callsign indicating the net or group name. The server's callsign is included in the digipeater callsign field. The H-bit is set. The users operate their TNCs with MCON ON and the LCALLS set to the net or group name. The server will prefix all user data with the sender's callsign and a sequence number. No active retransmission recovery will be provided. This is not a major problem because reliability problems are usually found on the uplink path. The remaining errors will be easily detected by any station observing a gap in the numeric sequence. A request for re-transmission may be manually solicited from the source. Future versions of this roundtable manager will interoperate with the COSI-Term and provide automatic recovery procedures.

Another service offered by the server will be distributlon services for alerting and bulletins. This is similar to the roundtable manager, but is single user. This system provides an error checked connection for uplink data and a broadcast UI-frame with the user data left unchanged. The source callsign is shown to be that of the orginator. The digipeater field is contains the callsign of the server. The H-bit is set.

COSI-Stack

COSI-Stack is the heart of each of these systems. The "stack" is a collection of OSI protocols which, provide communications services to users/user-applications. COSI-Stack is the software implementation of the OSI communications protocols outlined above. The software used in these servers is written in the "C" language for machines based on the Zilog Z-80 and Intel 8088 family of processors. The portability of this language allows the re-use code written for a particular system. We also derive the benefit of compatibility; when our source is upgraded it is incorporated into all the systems with a simple compilation of the changed module and re-linking for the target machine. This is extremely important because the maintenance of compatibility among multiple systems could be a chore if many machine-specific changes were required on each system.

COSI-Term

This is a basic user package for MS-DOS systems. Its major features include:

Automatic reconnect and resume;

Binary transfer;

Simultaneous file send/receive;

Simultaneous keyboard and file transfers;

ROSE and RTP support.

The package will be incorporated into the PMBx package.

SOFTWARE

We have a variety of systems already in use and development is continuing. The pace of software development is more rapid. Some systems are already available, some will be distributed around the time of the conference, and still others during the fall. Most source code is available and included with the executable modules. Those modules not currently available in source will either be made available at a later date, or will be replaced by modules whose source can be distributed.

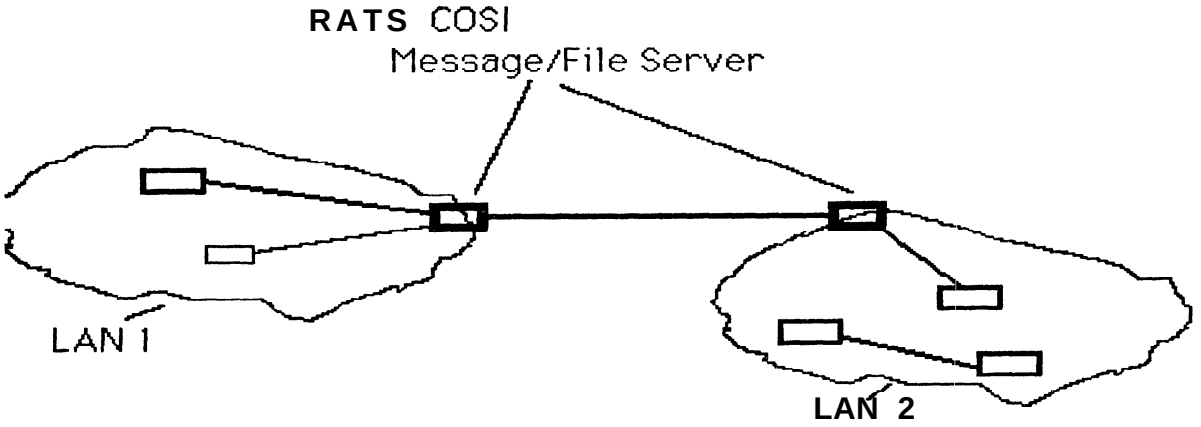
All modules are copyrighted, but are distributed for non-commercial use. Commercial use is possible, upon the conclusion of a licensing agreement. The agreements are designed to help provide the resources for enhancing the Amateur Packet Network. Organizations contemplating commercial use may consider offering products or services in lieu of funds. We all earn a decent living; we aren't interested in financing offices, secretaries, trips, etc. WE'RE BUILDING A NETWORK !

DOCUMENTATION

The Radio Amateur Telecommunications Society recognizes that OSI protocol expertise has been sparse, and therefore READABLE documentation virtually non-existent. We have begun to document the protocols, the implementation agreements/details and some system functional requirements. These documents comprise the RATS COSI Implementer's Guide. Interest in our work has been received through professional channels so the guide will be circulated in both the Amateur and professional communities. The Amateur community should take note. The Guide will evolve over time and updates and additions will be circulated via the nets.

SUMMARY

The Radio Amateur Telecommunications Society is dedicated to the development of OSI protocols and networks. We have addressed the basic operational modees through the application of a consistent architectural implementation: Open Systems Interconnection Protocols. Knowledge of OSI protocols is a relatively new expertise and it has taken the past few years build up interest, and expertise. We are now turning these into useful tools for the Amateur Community.

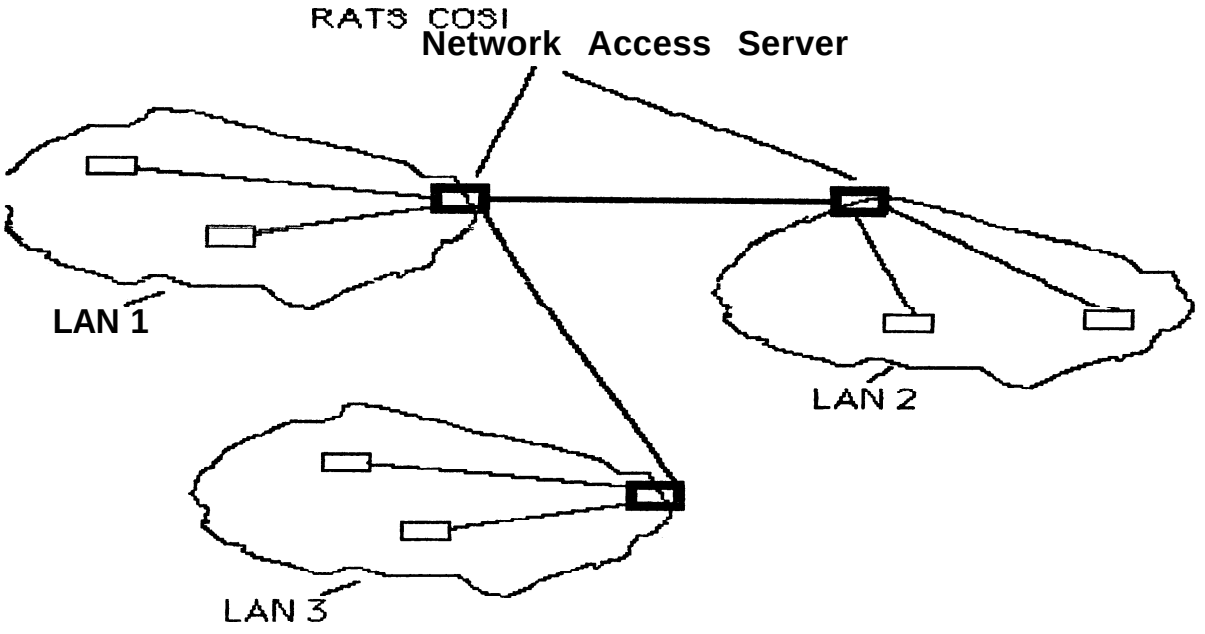


Message/File Server

PMBx User

Keyboard User

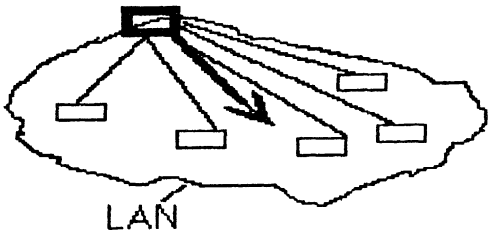
Application	**ROSE, RTP, ASN. 1**	**7**
Presentation	**X.226**	**6**
Session	**X.225** BCS	5
Transport	**X.224 TP-1**	4
Network	**x.25 PLP**	**3**
Link	X.25 HDLC/AFT	**2**
Physical	RS-232/V.35	1



Network Access Server

Keyboard User

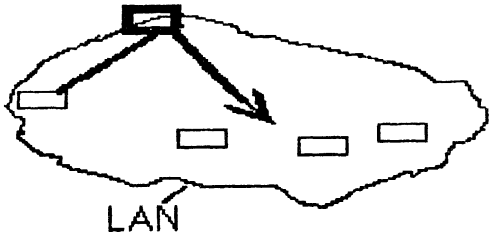
RATS COSI
Roundtable Server



Network Access Server

Keyboard or Computer User

RATS COSI
Broadcast Server



Network Access Server

Computer User

A HIGH PERFORMANCE PACKET SWITCH

Mike Brock, WB6HHV
Franklin Antonio, N6NKF
Tom Lafleur, KA6IQA
PO Box 9045
La Jolla, CA 92038

This paper describes the PS-186™, a high-speed multiport packet switch designed to support data rates up to 1 Mbit/second. It can also be used as a local packet node, mail system, or a gateway to other networks. It operates on 2.1 Watts of power. The PS-186 was designed by members of the San Diego Packet Radio Association (SANDPAC).

WHY WE BUILT IT

As requirements grow, existing packet radio hardware is running out of steam.

We believe the advancement of amateur packet radio requires packet switch hardware which will support data rates far in excess of today's standard 1200 bit/second links. While 1200 bit/second easily supports one typed conversation between two individuals, packet radio channels are shared channels, and must support many users simultaneously. To operate without congestion, the channel data rate must be many times higher than the rate desired by an individual user. Growing computer-to-computer communication is also creating a demand for higher user data rates. A more reasonable data rate for local area channels is 56 kb/s, with much higher rates, possibly 1 Mb/s, on the backbone.

A typical mountaintop requirement might be two 1 Mb/s ports (for point-to-point links to other mountaintops, say one North, one South), a 56 kb/s local radio port (connectivity to individual users), and a 1200 b/s local radio port (for connectivity with the TNC 1&2 community). With such applications in mind, we set forth to build a four-port packet switch capable of operation to 1 Mb/s.

DESIGN CONSIDERATIONS

The switch should provide a very flexible modem interface. Compatibility with existing modem designs was a must, but the modems we hope to use with this device haven't been designed yet! (There are promising developments, such as the WA4DSY *56* kb/s modem.) We don't know what data rates will become standard, and some modems will want to generate their own clocks, so the ports must have external clock options. We wanted to support modems that derive their own RX bit timing as well as those that do not, async as well as HDLC link-level protocols, etc.

Processors on existing packet hardware are running out of address space to hold sophisticated protocol software, and never really had the capacity to handle high data rate operations. We wanted a processor with more address space, more speed, and a large installed base, so that as many people as possible would have access to development tools. Because remote packet switch locations might be battery operated, the processor and all support circuitry should be available in CMOS versions. The solution was an Intel 80186 processor, running at 8 MHz[1], with no wait states, and full DMA[2] support. This processor, except for low-level I/O functions accessed only by I/O drivers, is software compatible with the IBM PC, IBM AT, and their clones, which now have an installed base of over *IO million units.* More sophisticated processors were considered, but none are available at as low a cost, or require as few support chips as the 186.

We felt that developers of complex packet-switch software should have the capability of loading new software into packet switches remotely. This would allow bug-fixes or enhancements to be installed in the network without PROM burning and mountain climbing. Painless software upgrading is essential if

[1]The board is designed to support 12 MHz 80186 chips. However, because the NEC DMA chips are not yet available in 10 or 12 MHz versions, initial boards will use 8 MHz processors.

[2]DMA stands for Direct Memory Access. It is a technique where sequential I/O operations are carried out by hardware in order to reduce processor load.

we want the networks to evolve. This requires that the switch contain a large amount of RAM, securely battery backed, and a secure mechanism for resetting the hardware remotely to initiate reload. (Amateur satellites have done this for years.) Some developers might not agree with this philosophy, so the board should contain massive amounts of ROM as well. We decided to design the board so that the processor's full 1 Mbyte address space could be populated with a variable mix of RAM and ROM. This required a design using 128Kx8 RAM chips that aren't quite available yet, so the design accommodates the use of smaller RAMs, with a corresponding temporary RAM limit of 256 Kbytes. We assume that prices of large static RAMs will continue to fall, but we added a bus expansion port capable of supporting an external RAM board just in case.

We questioned whether some applications might need more radio ports and if there should be a disk drive interface to support stand-alone development. Adding both the bus expansion port and a Small Computer System Interface (SCSI) port was the answer. The SCSI port can be used to interconnect several PS-186 boards, future expansion boards, a TAPR NNC (which also has a SCSI port), disk drives, etc.

Most existing TNCs generate significant RFI. A key issue was, therefore, to design a packet switch with RF-compatibility in mind. Because this board has more interfaces than most, there are many opportunities to introduce RFI problems, so we took special care. We decided that all digital circuitry should be above a complete ground plane — implying a multilayer circuit board.

As the feature list grew, affordability became a major consideration. We were afraid that we might be designing something so expensive that most hams couldn't afford one. Our solution was to make the board completely modular. This required considerable additional design time, and lots of option jumpers. Almost every chip on the board is now optional. Only users who require high-speed operation need install the DMA chips. Only users who require a SCSI port need install the SCSI chip. Ditto the real-time-clock chip, etc. Lower cost NMOS chips can be used in place of CMOS devices in most cases. Peripheral chips slower than the processor can optionally be used by running them on a ÷2 clock (with reduced performance of course), and on and on.

Finally, we hoped it would all fit on one board, and it did!

SOFTWARE CONSIDERATIONS

This switch is intended to serve as a hardware host for all the evolving packet radio protocols. We're not taking sides in the protocol wars. We expect that the developers of COSI, TEXNET, KA9Q's TCP/IP, and NET/ROM will all be able to port their software to the PS-186. As described in a companion paper, Brian Kantor, WB6CYT, is developing a multitasking operating system kernel for the PS-186, which may form the basis for future software developments.

WHAT WE BUILT

A specification summary of what evolved as the PS-186 follows. As you can see from this list, the PS-186 is designed to take advantage of many high-technology chips. Several other high-tech parts are not directly apparent. For example: power control for the battery-backed RAMs is simplified by the use of Dallas Semiconductor nonvolatile controllers, and PALs[3] are used extensively to replace what would otherwise be very complex random logic. Without PALs, the board would have been *twice as large.* The combination of advanced technology and planning for future developments, especially in the areas of faster chips and larger RAMs, should allow the PS-186 a long life.

Processor: Intel 80186 @ 8 MHz
- Option for 10 or 12 MHz CPU
- Option for CMOS 80C186 (same speeds)

PROM: 16K to 256K bytes
- 2764 thru 271024 byte-wide EPROMs supported
- 28 or 32 pin devices
- Optional wait states

RAM: 16K to 1023K bytes
- Up to eight of' 6264, 62256, or 621024 supported
- 28 or 32 pin devices
- Battery backup optional for each pair of RAMs
- No wait states

[3]PALs are Programmable Array Logic chips.

Serial Channels: 2 or 4 ports
- 1 or 2 Zilog 8530A or CMOS 85C30
- Simultaneous full-duplex DMA on all 4 ports
- Sync or Async, NRZ or NRZI or bi-phase
- 1X, 16X, or 32X clocks
- Hardware capable of over 1 Mbit/second on all channels
- All basic signals: TXD, RXD, RXclk, RTS, CTS, outbound TXclk, inbound TXclk, DTR, CD
- TTL voltage levels
- Small daughter boards per serial channel for level conversion or small modems

SCSI Port
- NCR 5380 or CMOS NCR 53C80 chip
- SCSI-Plus, initiator or target, ANSI X3T9.2 Compatible

Interrupt Controller
- Intel 8259A
- Wired for optional RMX-86 compatibility

DMA: 10 simultaneous DMA channels
- 2 channels inside 80186 -- used to support SCSI port
- 2 NEC 71071 chips provide 8 channels for serial ports

Time-of-Day Clock
- National DP857 1 chip
- Battery backed

Reset Circuits
- Power-on reset
- Manual reset
- Watchdog timeout reset
- Remote-reset circuit provides over-the-air reset

General Purpose I/O
- Bus expansion port w/DMA & interrupt support
- 4 bit option jumper
- 4 uncommitted LEDs

Power
- 5 Volt only operation
- 1.6 Amps using NMOS & ALS-TTL (i.e., 8 Watts)
- 420 mA using all CMOS parts (i.e., only 2.1 Watts!)
- 12 Volt optional distributed to daughter boards only

Board
- 7.75" x 11.5" (same size as two $5\frac{1}{4}$" disk drives)
- 6-layer construction (GND, VCC, 4 signal layers)

SOME OBVIOUS QUESTIONS

Q: Can all those ports *really go* that fast simultaneously? Won't they overrun? Can memory keep up?

A: DMA operations use 4.5 clock cycles on average. In an 8 MHz machine, that means there can be up to 1.8 million byte DMA accesses/second. All 4 serial ports running 1 Mbit/sec RX & TX at the same time generate only 1 million bytes/second and would therefore consume a maximum of 56% of the memory cycles. Running four *full-duplex* ports at 1 Mbit/sec is an extremely aggressive configuration, and you would seldom expect all 8 channels to be active simultaneously, but even when they are, about half of the memory cycles are still available for the processor.

Q: Is the 186 processor *really* capable of keeping up with greater than 1 Mbit/second *throughput?*

A: It depends on how carefully the software is written. DMA I/O relieves the processor of the burden of handling each character, so that the processor now need only respond to interrupts at the end of every received and transmitted packet, and perform the protocol processing required by each received packet. Consider a configuration with two 1 Mbit/sec full-duplex ports, such as described on page 1. Assume a packet size of 512 bytes, and both ports saturated with traffic. At 1 Mbit/sec, that's one packet arriving every 2 milliseconds. First we look at how many instructions the 186 can execute in that time. On average, the processor's speed is determined by memory access. A word access uses 4 clock cycles, and an instruction requires on the average about 2 word accesses. So in an 8 MHz system, the 186 runs at about 1 million instructions per second (MIPS). Processor speed is reduced to about 0.7 MIPS, because DMA activity in this example may require as many as 28% of the memory cycles. Finally, in 2 ms of running at 0.7 MIPS, the 186 can execute about 1400 instructions. That's enough for carefully written software to process the average packet and send it on its way. Note that packet size is an important parameter on high data rate links, because it determines the maximum packet arrival rate, and therefore processor throughput.

Q: Is it really necessary to build your own custom processor board in 1987? Why not use a personal computer or clone board?

A: Existing PCs and clones don't have the required I/O capacity. They aren't battery backed. They draw too much power. IBM PC system board clones, which are attractively priced, have only byte-wide memory, hence lack the processor speed or memory

cycles to support the I/O we have in mind. They do contain three DMA channels, but design constraints keep them from being used simultaneously. The IBM AT system board comes closer, but it still has some awkward DMA limitations, it would require the addition of a fairly complex. I/O board, it still wouldn't be battery backed, and would draw too much power for many applications. The MAC II has the same problems at a higher price. Personal computers weren't designed to be packet switches.

Q: Why no power supply on the board?

A: We didn't want to put a series-pass regulator for 12V operation on the board because it would dissipate over 14 Watts in the worst case (fully loaded NMOS). We didn't want to design in a switching power supply because we aren't power-supply engineers. Besides, this board is designed to be part of a system. There will be some modems also needing power. A single external power supply for the PS-186 and modems seemed a reasonable solution.

Q: Why TTL I/O? Why no RS-232?

A: Most of the modems we had in mind use or will use TTL I/O. Others may use V.35 or RS-422,. RS-232 interfaces are the exception rather than the rule at data rates above 19.2 kb/s. Each serial port provides a connector for a small daughter board which can con tain level translators, and appropriate connectors. A daughter board which converts a port to RS-232 has been designed for those applications requiring RS-232.

DETAILED CIRCUIT DESCRIPTIONS

The following describes the PS-186 hardware in considerable detail. Casual readers may wish to skip to the next section. It will be useful to refer to the block diagram and schematic. Reference designators refer to the schematic.

80186 Microprocessor

The Intel 80186 microprocessor at U12 provides the central core of the PS-186. This processor is capable of directly addressing up to 1 Mbyte of memory and 64 kbytes of I/O. The 80186 has many useful features (on-chip address decoding, wait-state generation, DMA controllers, and counters), and we use most of them. The on-chip memory and I/O decoding simplifies and reduces the logic. One of the two DMA channels is used to support an optional SCSI port and the timers are used for various software functions. Unused CPU features (one DMA channel, timer/counter inputs, etc.) are accessible at W16, W20, El, and E2.

The processor is buffered by chips U8, U9, U13, U14, and U27. This improves circuit loading characteristics, provides isolation, and demultiplexes the processor's address and data lines. U31 buffers the read and write lines and separates them into memory read, memory write, I/O read, and I/O write. The I/O write signal is delayed by flip-flop U30 to provide data setup time for peripherals.

Clock speed configuration jumpers W21 and W25 permit the use of various speed parts. The top frequency of the board (2x CPU clock) is established by hybrid oscillator Y 1.

EPROMs

Two byte-wide EPROM sockets are provided at U2 and U17. Each EPROM supplies one byte of a 16-bit word. Address configuration jumpers W2, W3, and W18 specify 16K x 8 chips (2764s) up to 128K x 8 chips (271024s). The maximum amount of EPROM is 256 kbytes.

The EPROMs reside at the very top of the CPU memory space, which is where the processor starts executing after it wakes up from a reset. Software configured EPROM sizing determines how far down in memory EPROM extends., This dynamic mapping of high memory allows, for example, programs to map in EPROM, read some data tables, and then map RAM back in. This scheme makes possible memory configurations greater than 1 Mbyte, even though the processor can only see 1 Mbyte at any time. The minimum software configurable EPROM is 1 kbyte.

The CPU, under program control, can insert up to three wait states allowing the use of almost any speed EPROM.

RAM

The PS-186 has space for up to four pairs of byte wide static RAMs (eight devices) at U3 & U18, U4 & U19, U5 & U20, U6 & U21. Three sizes of RAM chip can be used, 8K x 8 (6264), 32K x 8 (62256), or 128K x 8 (621024), selected by W5, W27, and W28. A board configured with eight 128K x 8 RAMs uses the full 1 Mbyte CPU address space, with some of the top-most locations lost to EPROM as previously I;";&--cussed.

RAM is optionally battery backed. Power for each pair of RAM chips is individually configured for Vcc

or a battery source by W4, W6, W7, and W8. The optional battery with all the switching and isolation required is included on the board. U36 provides the final section of RAM decoding and power isolation when a battery is used. A lower cost chip at U35 does this if battery backup is not required. The RAM isolation chip can switch up to 40 µA of backup power to the RAM. If the total RAM standby current exceeds this, an optional power control chip can be installed at U44 and enabled by jumper W33. W15 is used if there is no battery in the system. When deciding whether to include a battery, keep in mind that the time-of-day clock requires a battery if it is to continue timekeeping during power failures.

PAL U28 determines how much memory space the processor has allocated to EPROM and prevents the CPU from accessing RAM in that address range. It steers the memory write signal so byte writes don't disturb the other half of the word, and uses the RAM size configuration to determine which RAM pair to enable.

Serial Channels

Two Zilog 8530s at U54 and U55 provide four full duplex serial channels with modem control. The 8530s support synchronous and asynchronous serial line formats including support of bit and byte oriented synchronous protocols. These chips include baud rate generators as well as phase lock loop clock recovery circuits. Clock option jumpers allow the use of any speed 8530.

One of the unique features of the PS-186 is the 8530 control circuitry. This circuit takes care of the special timing requirements of the 8530s, eliminating the need for special programming, and allowing full use of DMA. The circuitry optimizes system performance by overlapping 8530 valid access recovery times and blocking DMA requests from individual 8530s while they are recovering. This maximizes the time available to the CPU. It also maximizes 8530 access since one 8530 can be serviced while the other is recovering. Recovery time is configured by W39, which specifies time constants for the two timing counters (U50 and U51). The counters are controlled by PAL U53 which performs timing optimization. The PAL also combines the read, write, and reset signals for the 8530s.

The 8530 interrupt acknowledge is synchronized by the flip-flops at U33. This allows full use of the 8530's interrupt vectors.

An extensive set of serial clock options is provided for each channel. W19, W22, W23, and W24 configure the outgoing transmit clock, incoming transmit clock, and incoming receive clock for channels A, B, C, and D, respectively. Provisions are made for 32x, 16x and lx clocks. Options specifically allow for a 32x or 16x baud rate clock from the 8530 to be divided down to lx externally and sent back to the 8530, making it possible to receive using the internal phase lock loop while maintaining a steady transmit rate.

The serial port connectors (J5, J6, J7, and J8 for channels A through D, respectively) are 16-pin header strips. All of the signals on these connectors are TTL. The board is layed out to accept small daughter boards which are large enough to contain an RS-232 interface, a TAPR modem disconnect interface, or even a full 1200 baud 202 modem. Filtered +5V and +12V power is supplied to each connector. Signals supported on the connector are: transmit data, receive data, RTS, CTS, DTR, DCD, incoming transmit clock, outgoing transmit clock, and incoming receive clock.

A byte multiplexer formed by U10, U15, and controlled by PAL U34, lets the 8530s respond to byte addresses. The multiplexer allows DMA operations with the 8530s to access sequential byte locations. The PAL also picks up some housekeeping functions to reduce part count.

DMA Channels

There are a total of IO *DMA channels* on the PS-186. Two of them are embedded in the processor. The first processor DMA channel supports the SCSI interface. The second is not associated with any device and may be used for memory-to-memory transfers.

The remaining eight DMA channels are provided by two NEC 7 1071 DMA chips and are dedicated to the serial channels. Two DMA channels are provided for each serial channel, resulting in full-duplex DMA operation. The DMA chips are capable of generating any RAM address without concern for address boundaries. The DMA chips do not see EPROM, and they always address RAM regardless of the processor memory map.

The DMA circuitry uses the fly-by mode of operation. This means that memory addresses are generated, I/O addresses are implied, and controls for both are activated simultaneously. During this operation the data is not stored anywhere, it just "flys by" on the bus, hence the name. Fly-by allows each byte DMA operation to complete in an average of 4.5 clock cycles. At 8 MHz,

this means over 1.7 million bytes/second, (i.e., over 14 Mbits/second) of DMA I/O are possible. Jumpers W9, W10, W11, W13, W30, and W31 permit either or both DMA chips to be removed.

Arbitration between DMA chips is provided by PAL U7. The PAL determines which controller will get the internal bus when a conflict. arises. It also arbitrates bus requests from the expansion port. Jumper W12 selects fixed or round-robin priority for the three requestors.

DMA chip interrupts are latched by U59 before being sent to the interrupt controller. These flip-flops are cleared by accessing specific I/O locations.

DMA interface to the 8530s is provided by PAL U52 and flip-flops U37 and U39. The PAL provides the implied I/O address required for fly-by DMA. The flip-flops provide a way to shut off the 8530 DMA requests once service has started, and an alternate way to start a transmit data stream. Accessing the I/O port associated with each flip-flop generates the first request to the DMA controller. This eliminates the need for the processor to send the first character of each DMA transfer directly to the 8530.

SCSI Port

To permit connection to other devices (TAPR NNCs, serial expansion boards, disk drives, or even other PS-186s) the PS-186 includes an optional NCR 5380 SCSI controller at U25. If the physically larger CMOS version of this chip is preferred it plugs in at U24. DMA channel 0 of the processor supports the SCSI port. The 220/330 ohm SCSI termination is provided by RN3 and RN4. W14 selects an internal or external power source for the termination. The internal power source is filtered and fused before it supplies any termination power, internal or external. The external SCSI connector, J3, is a standard 50-pin header.

Support for the SCSI chip is provided by the PIO (8255) at U43. Port A of the PIO is connected to 8 jumpers (W32) which set the SCSI address. The combination of an eight bit address and the 5380 chip permits use of the SCSI-Plus enhancements.

Interrupt Controller

Interrupts are handled by both the processor and an Intel 8259 at U40. The processor accepts interrupts and vectors from the 8530s, and by means of cascading, the 8259. The 8259 accepts interrupts from the DMA chips, SCSI chip, time-of-day clock, and bus expansion connector.

Time-of-Day Clock

A National DP8571 at U56 provides full clock calendar functions for the PS-186. The chip generates two distinct interrupts based on various alarm and timing functions. If a battery is supplied, the chip will continue timekeeping during power outages. Trim capacitor C61 adjusts the clock timebase.

PIO

An Intel 8255 PIO at U43 to provides a number of services to the board. PIO port A supplies the SCSI port address as discussed above. Port B is connected to W37, a four position jumper block set aside for software options. The other four bits of port B read the reset status of the board. By reading these, the software can determine the source of a hardware reset (power-on, manual, watchdog, remote). Port C is an output port divided into two halves. The first half drives the DTR lines on each serial port connector. The other half drives four independent LEDs which are used by the software to indicate program status or other conditions. W1 removes power from the LEDs when they are not in use,

Watchdog Timer

The reset control chip at U1 provides three services. The first is a watchdog function. If the watchdog I/O location is not accessed at least once every second, Ul generates a hardware reset. W40 enables or disables the watchdog. The second function is a reset switch debounce and sequencer. When switch Sl is pressed or contacts closed on de'bug connector J2, the board is reset and held that way for 250 ms after the switch is released. The third function is to hold the board in a reset condition when Vcc is out of tolerance and for 250 ms after the power stabilizes. This restrains the CPU during power transients

Remote Reset Circuit

This circuit provides a remote processor reset capability. The 4th radio port, channel D, is monitored for the presence of a 255 bit long pattern (PN^4 sequence). These patterns are long enough so that they will never occur accidentally in random data (an unsquelched rnodem input may produce continuous random data). They also contain sequences of l's which violate HDLC bit-stuffing rules, so that they cannot occur in valid HDLC frames.

---

[4] PN sequences are also sometimes called "maximal-length-shift-register-sequences". They are patterns of l's and O's that are easy to generate, and have nice statistical properties.

During normal operation, the *8530* SCC chips do bit synchronization for incoming data. Unfortunately, the reset circuit must operate assuming the processor has crashed, so cannot make use of the 8530s. PAL U48 is an optional bit-timing recovery state-machine. Jumper W35 selects NRZ or NRZI output, or bypasses U48 entirely for those modems that do their own bit-synchronization. Jumper W34 selects between the recovered clock or an optional clock from the modem. PAL U46 searches the input data stream for the reset sequence.

The PAL has been programmed to search for one of 12 PN sequences[5] selected by jumper W38. We didn't want a reset command to accidentally trigger every PS-186 within hearing range of the sender. Our intention is that nearby PS-186 nodes will use different reset sequences, keeping them individually resettable.

When the reset sequence is detected, the circuit causes a non-maskable interrupt (also optional per jumper W29). This circuit interacts with the watchdog to initiate a timeout. If the software does not respond to the interrupt within a second, a hard processor reset is forced. The need for a remote hard reset is clear. For those who may want to implement a more sophisticated soft reset in software, this circuit provides that capability without sacrificing the hard reset needed when the processor is severely crashed.

Board Power

Power is supplied to the board through a standard 4-pin disk drive style connector at J1. The board is expected to survive in noisy environments so attention was given to filtering and bypassing. As power enters the board it is filtered by ferrite beads and capacitors. At each individual serial port connector the power is again filtered and bypassed by ferrite beads and capacitors. The liberal use of bypass capacitors and a multilayer circuit board with power planes keep the noise and susceptibility down.

Both +5 and +12 Volts are brought in from the power connector but only +5V is used on the board. The +12V is supplied to the serial port connectors in case the daughter boards require it.

Bus Expansion Port

A 60-pin bus expansion connector, J4, provides expandability. J4 contains all the address, data, and control lines necessary to add external peripheral devices, bus request and acknowledge lines to support external DMA chips, two interrupt lines, and of course, power.

SCHEDULE & AVAILABILITY

The project began in a meeting between KB5MU, KA6IQA, WB6HHV in May of 1985, where the basic goals of the project were established. Design began in earnest a year later. A wirewrap prototype (see photo) was completed in July 1986, test code was running on the board in August, and the design was committed to printed circuit layout in November. PC layout was completed in July 1987. At times it seemed our progress gave new meaning to "slow". In retrospect, the only slow parts were "getting started" (an age-old schedule killer) and the board layout. We don't fault the people who donated their efforts for the 6-layer board layout. Charity work is necessarily low priority, so free services don't often fit any particular schedule.

The present schedule calls for ten assembled and tested development prototypes to be available in October. We feel lucky to have obtained donations from various manufacturers of most of the sophisticated integrated circuits required for the prototype build. We had to pay for boards, sockets, and many of the 'ordinary' ICs These protos will be made available to a (nedessarily limited) number of software developers who have demonstrated a capability to produce useful software products to the amateur radio community. The boards will come with test code and a debugger in EPROM, source on $5\frac{1}{4}$" IBM PC diskette, and a technical manual. We *hope* to make 2nd round production quantity boards available in January 1988. The distribution mechanism for production boards has not been worked out.

While we can make no guarantees about prices which may be offered by an eventual distributor, we hope bare boards will be made available for under $75. All parts on the PS-186 can be purchased in small quantities for $135. (for a minimum configuration) or $300. (maximum) from suppliers advertising in Byte magazine.

'There are 16 PN sequences of length 255, but we ran out of product terms in the PAL, so it only searches for 12 of them.

ACKNOWLEDGEMENTS

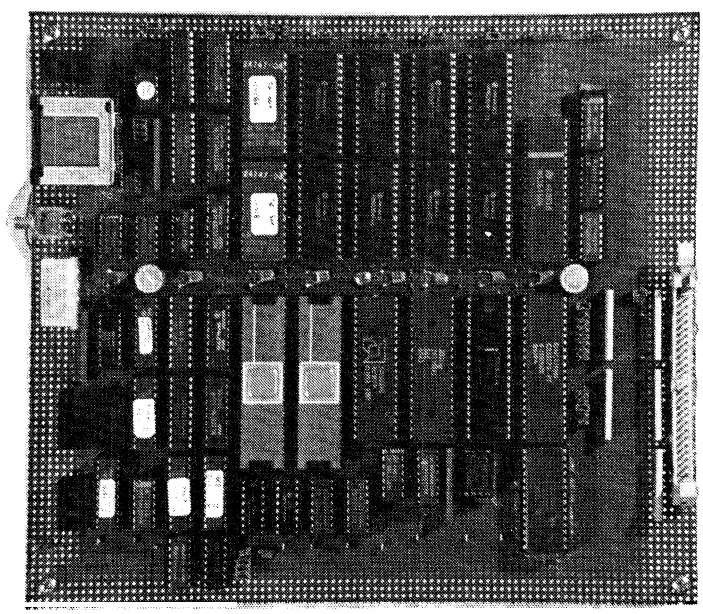
We would like to thank Texas Instruments, Motorola, Intel, NEC, C&K switch, NCR, 3M, Dallas Semiconductor, Siemens, Sony, AMD, Toshiba, Zilog, Panasonic, Varta, National Semiconductor, Bourns, Fujitsu, Statek and Hitachi, as well as their reps and distributors for their donations of parts for the prototype boards, and the CAD department of M/A–COM LINKABIT who donated PC board layout and artwork generation services. We would also like to thank Edye Brown for carefully reviewing this paper, and KB5MU and WB6EME for porting the debugger.

The authors can be reached by various electronic means:

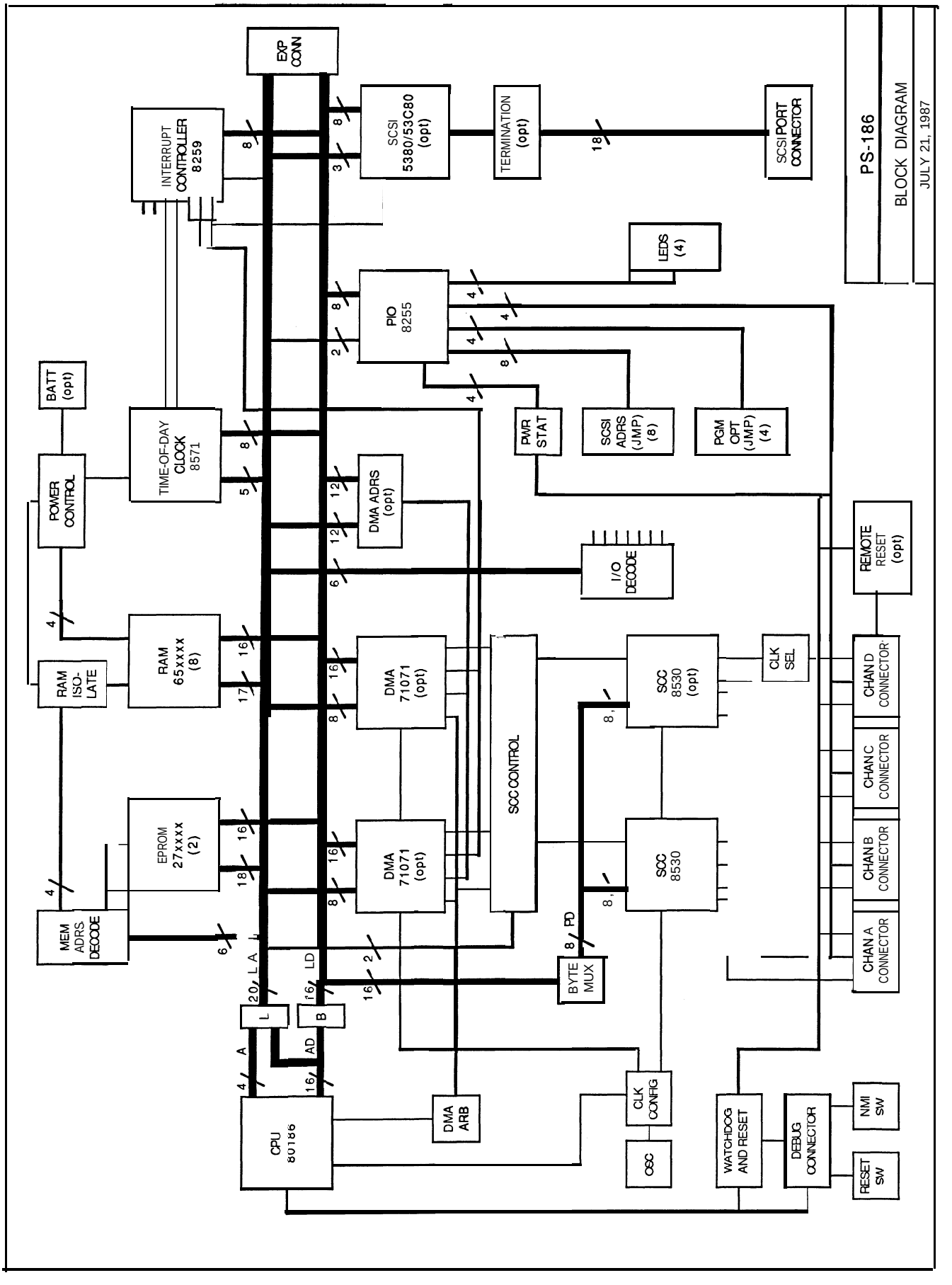
Call	CIS ID	Internet
N6NKF	76337,1365	..qualcomm@a.isi.edu
WB6HHV	... 76246,546	.WB6HHV@sdcsvax.ucsd.edu
KA6IQA	 76244,154	.lafleur@sdcsvax.ucsd.edu

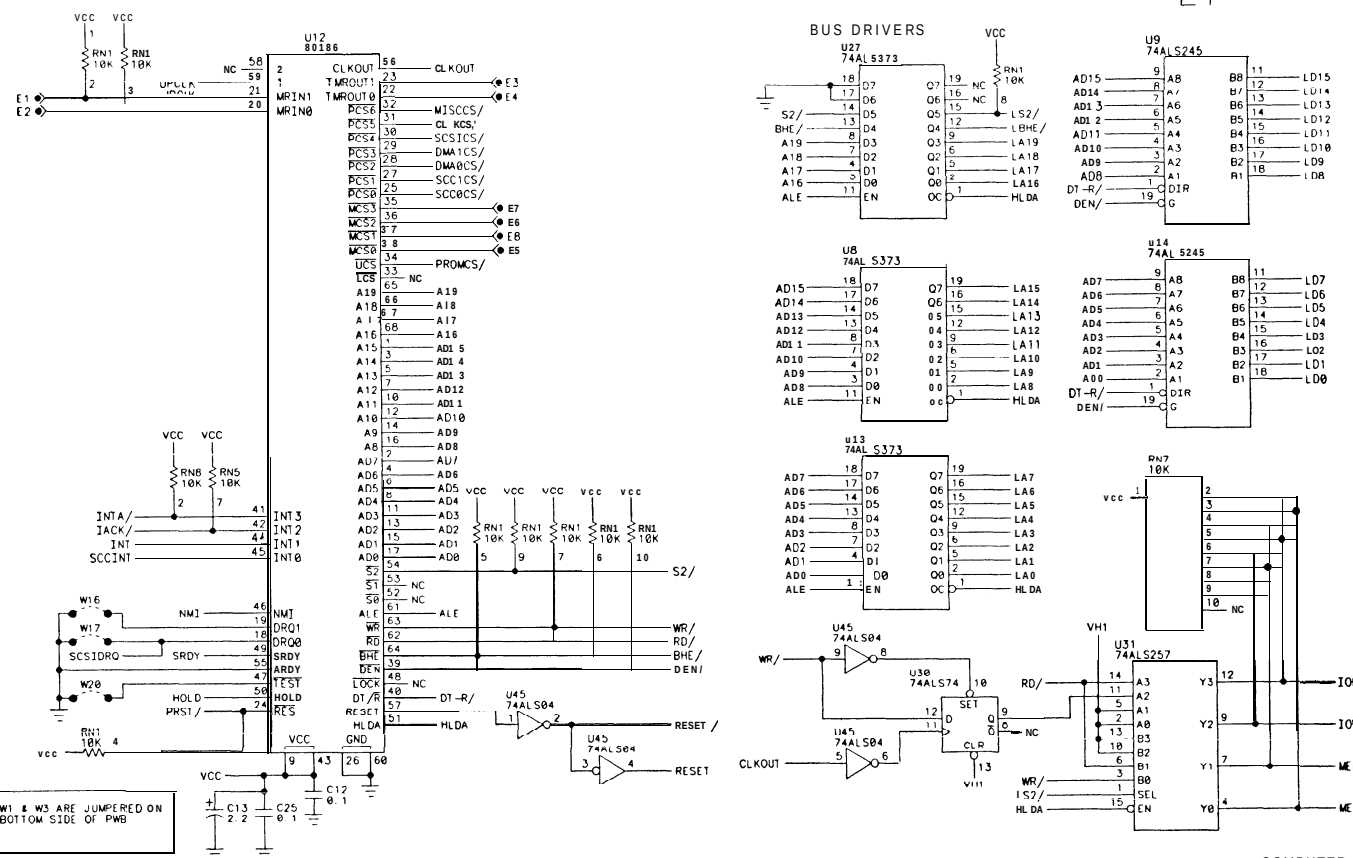
REFERENCES

1. Intel Embedded Controller Handbook (1987) [80186 Processor]
2. Intel 80C186 Target Specification (1986) [80C186 Processor]
2. Intel Microprocessor and Peripheral Handbook (1987),
   Volume 1 [8259 interrupt controller] ,
   Volume 2 [8255 PIO].
3. NEC Microcomputer Products Data Book (1987) Volume 2 [µPD71071 DMA Controller]
4. Zilog Components Data Book [8530 Serial I/O]
5. Dallas Semiconductor Product Data Book [DS1221, DS 1232, DS1259 RAM power control chips]
6. National Semiconductor DP857 1 Timer Clock Peripheral (TCP) Data Sheet
7. NCR 5380-53C80 SCSI Interface Chip Design Manual (3/86)



PS-186 Wirewrap Prototype


PS-186
BLOCK DIAGRAM
JULY 21, 1987
EXP CONN
INTERRUPT CONTROLLER 8259
SCSI 5380/53C80 (opt)
TERMINATION (opt)
SCSI PORT CONNECTOR
LEDS (4)
PIO 8255
BATT (opt)
TIME-OF-DAY CLOCK 8571
PWR STAT
SCSI ADRS (JMP) (8)
PGM OPT (JMP) (4)
POWER CONTROL
DMA ADRS (opt)
I/O DECODE
REMOTE RESET (opt)
RAM ISO-LATE
RAM 65xxxx (8)
DMA 71071 (opt)
SCC CONTROL
SCC 8530 (opt)
CLK SEL
CHAN D CONNECTOR
CHAN C CONNECTOR
EPROM 27xxxx (2)
DMA 71071 (opt)
SCC 8530
CHAN B CONNECTOR
MEM ADRS DECODE
BYTE MUX
CHAN A CONNECTOR
CPU 80186
DMA ARB
CLK CONFIG
OSC
WATCHDOG AND RESET
DEBUG CONNECTOR
NMI SW
RESET SW



5 INDUCTOR VALUES ARE IN MICROHENRIES

4 CAPACITOR VALUES ARE IN MICROFARADS

3 RESISTORS ARE IN OHMS. ±5%, 1/4 WATT

2 PART NUMBERS ARE ABBREVIATED AND FOR REFERENCE ONLY FOR COMPLETE PART NUMBERS, SEE PARTS LIST

1 INTERPRET DRAWING PER DOD-ST D-100

NOTES UNLESS OTHERWISE SPECIFIED

6 REFERENCE DOCUMENTS

ASSY DWG

PARTS LIST

REVISIONS			
RCV	DESCRIPTION	DATE	APPROVED
	RELEASE SEE DCR	-	-

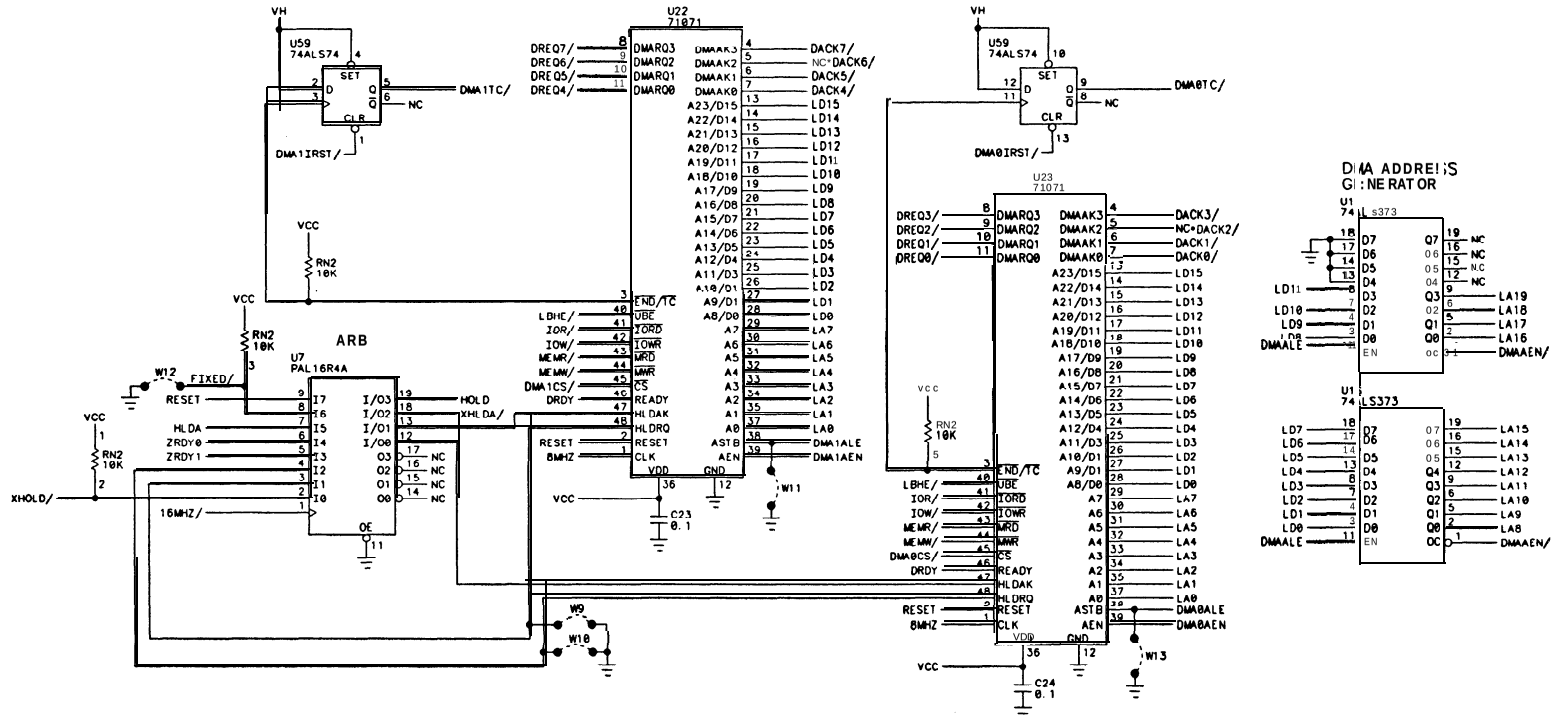
COMPUTER GENERATED DRAWING

	CONTRACT No	DATE	
	DRAWN		TITLE
	CHK		SCHEMAT J-C
	ENG		PS-1 86
	PROJ ENG		
	MFG		
	PTS CONT		
	LOG		SIZE B / CAGE NO / DRAWING NO LAFLEUR / REV
NEXT ASSY / USED ON	P A		
APPLICATION	RLSE		SCALE NONE / SHEET 1 OF

DWG NO	LAFLEUR	SHT 2

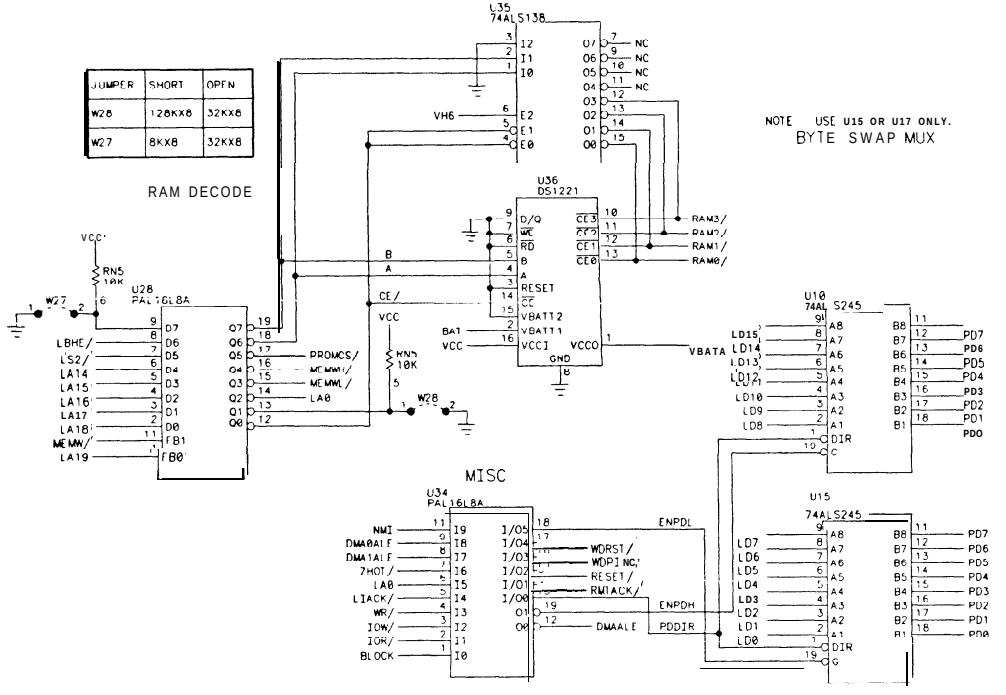
REVISIONS

REV	DESCRIPTION	DATE	APPROVED
–	SEE SHEET 1		



DMA

PS-186	SIZE B	CAGE NO	DRAWING NO LAFLEUR	REV –
	SCALE: NONE		SHEET 2	



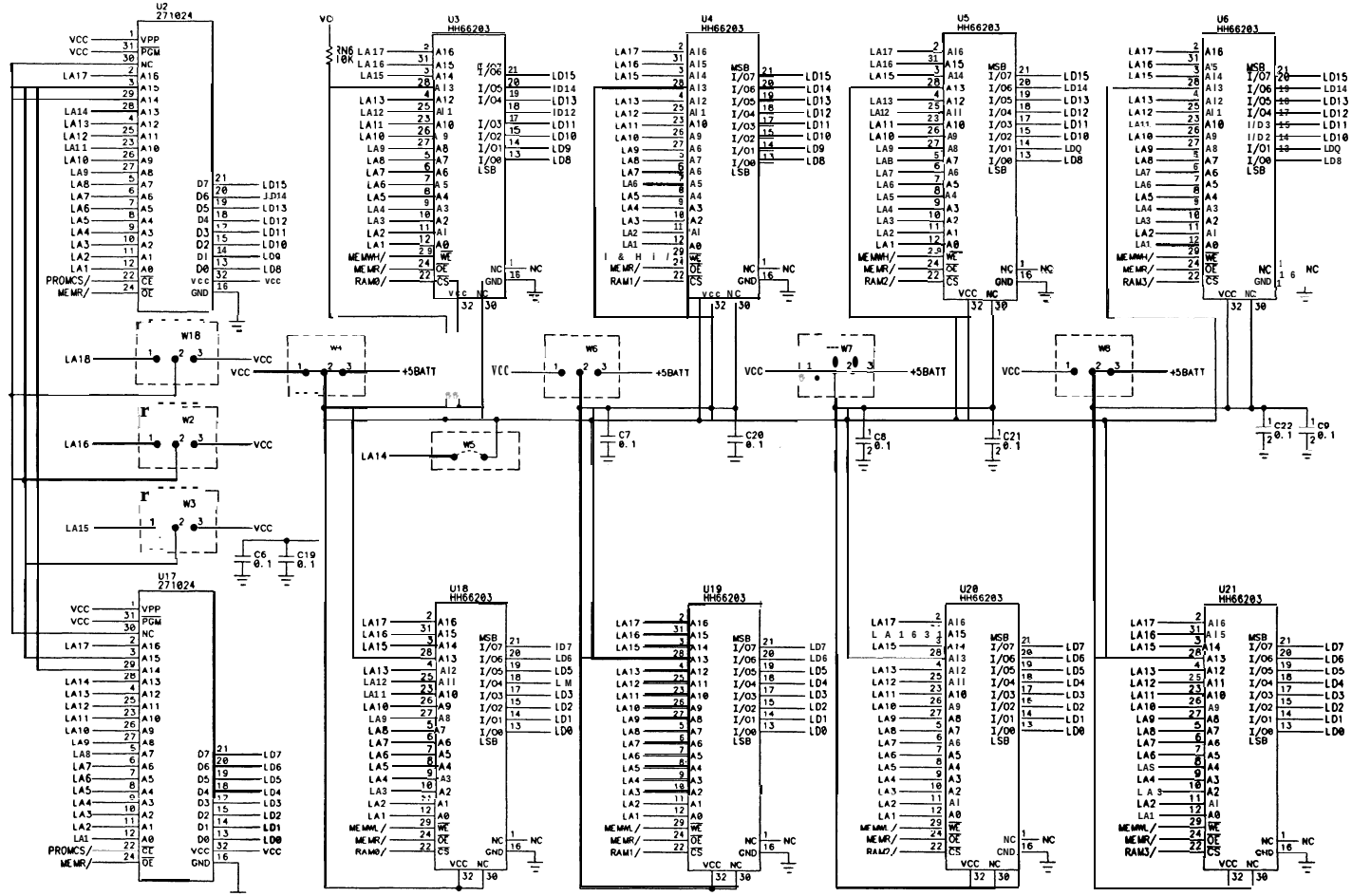
RAM DECODE
AND BYTE MUX

PS-186	SIZE B	CAGE NO	DRAWING NO LAFLEUR	REV -
	SCALE NONE		SHEET 3	

DWG NO	LAFLEUR	SHT 4

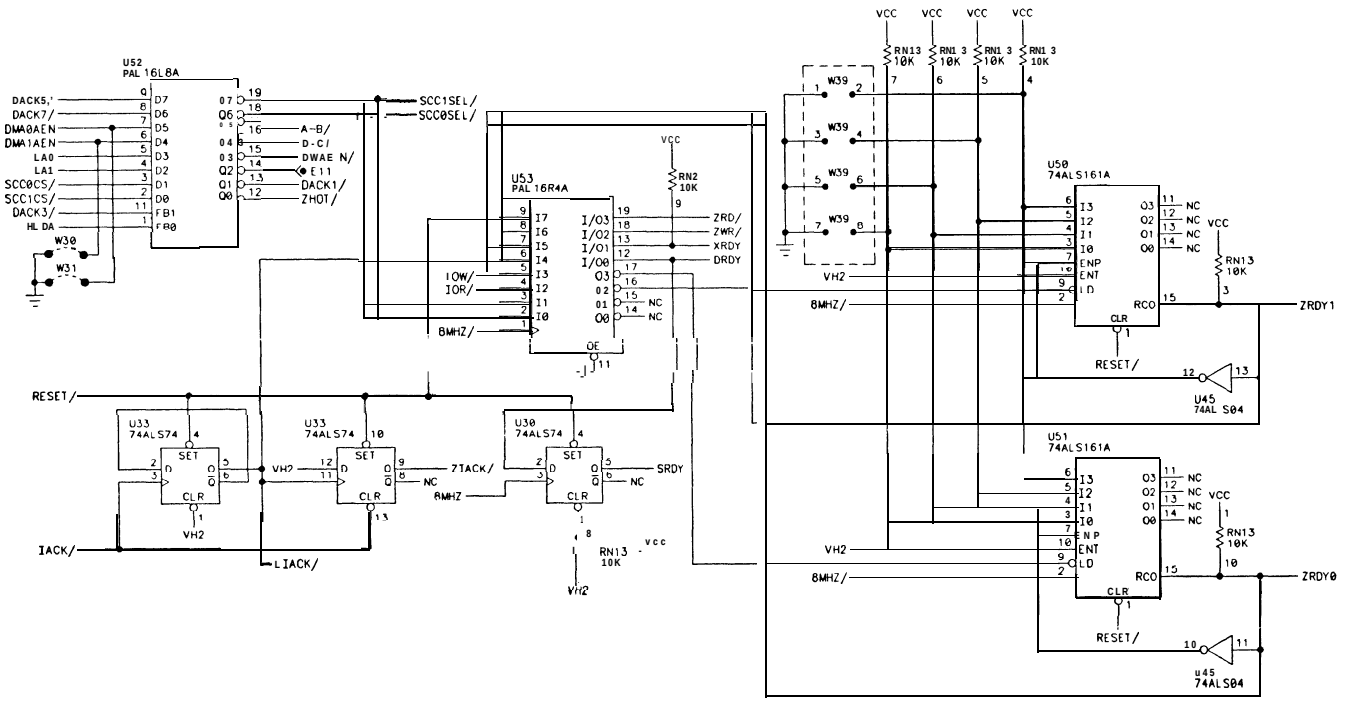
REVISIONS

REV	DESCRIPTION	DATE	APPROVED
-	SEE SHEET 1	-	-



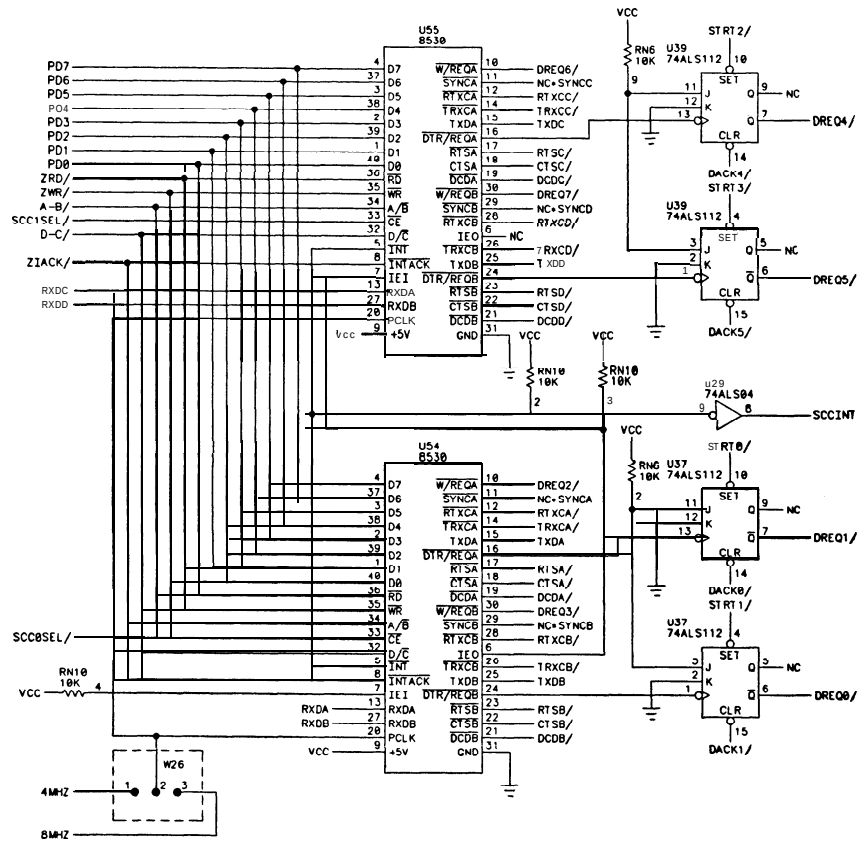
MEMORY ARRAY

PS-186	SIZE B	CAGE No	DRAWING NO LAFLEUR	REV -
	SCALE: NONE		SHEET 4	



SCC T IMING

Ps-186	SIZE B	CAGE NO	DRAWING NO LAFLEUR	REV -
	SCALE NONE		SHEET 5	



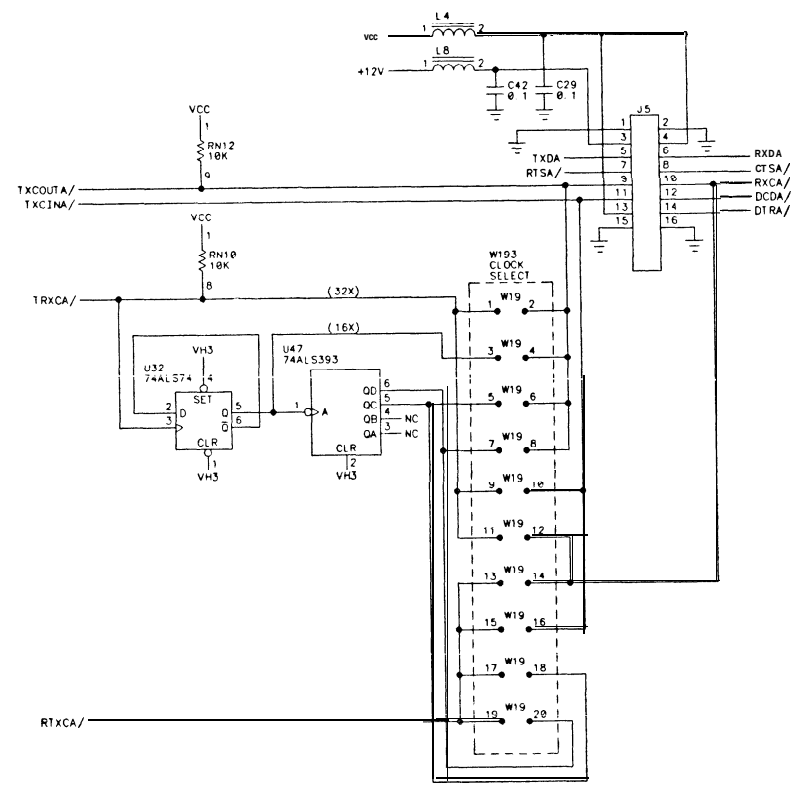
SCC

PS-186	SIZE B	CAGE No	DRAWING No LAFLEUR	REV –
	SCALE NONE		SHEET 6	

DWG NO LAFLEUR SHT 6

REVISIONS

REV	DESCRIPTION	DATE	APPROVED
–	SEE SHEET 1	–	–



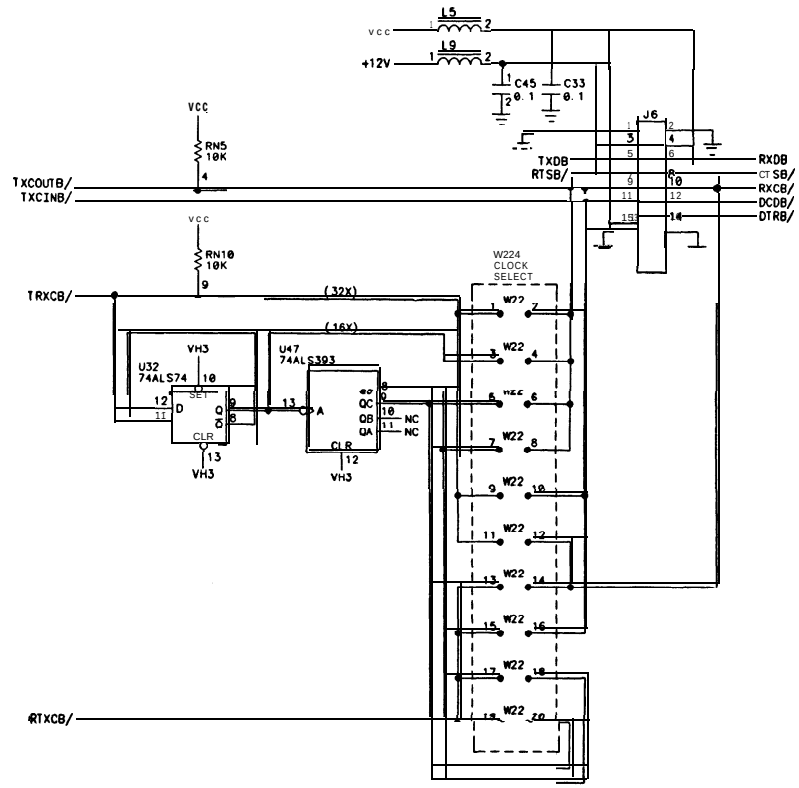
DTE CHANNEL A

PS-186	SIZE B	CAGE NO	DRAWING NO LAFLEUR	REV –
	SCALE NONE		SHEET 7	

DWG NO	SHT
LAFLEUR	8

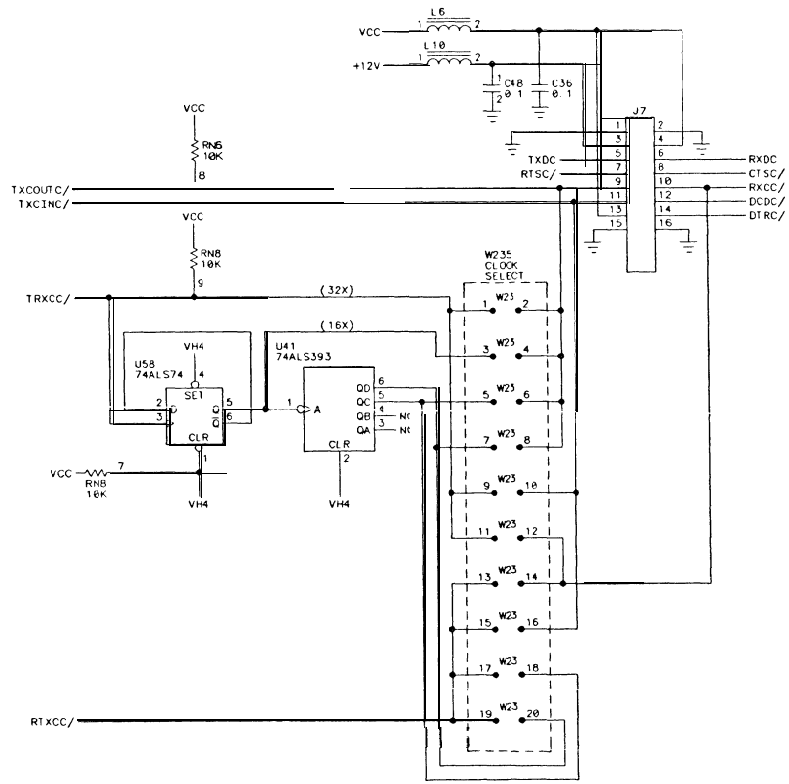
REVISIONS

REV	DESCRIPTION	DATE	APPROVED
-	SEE SHEET 1		



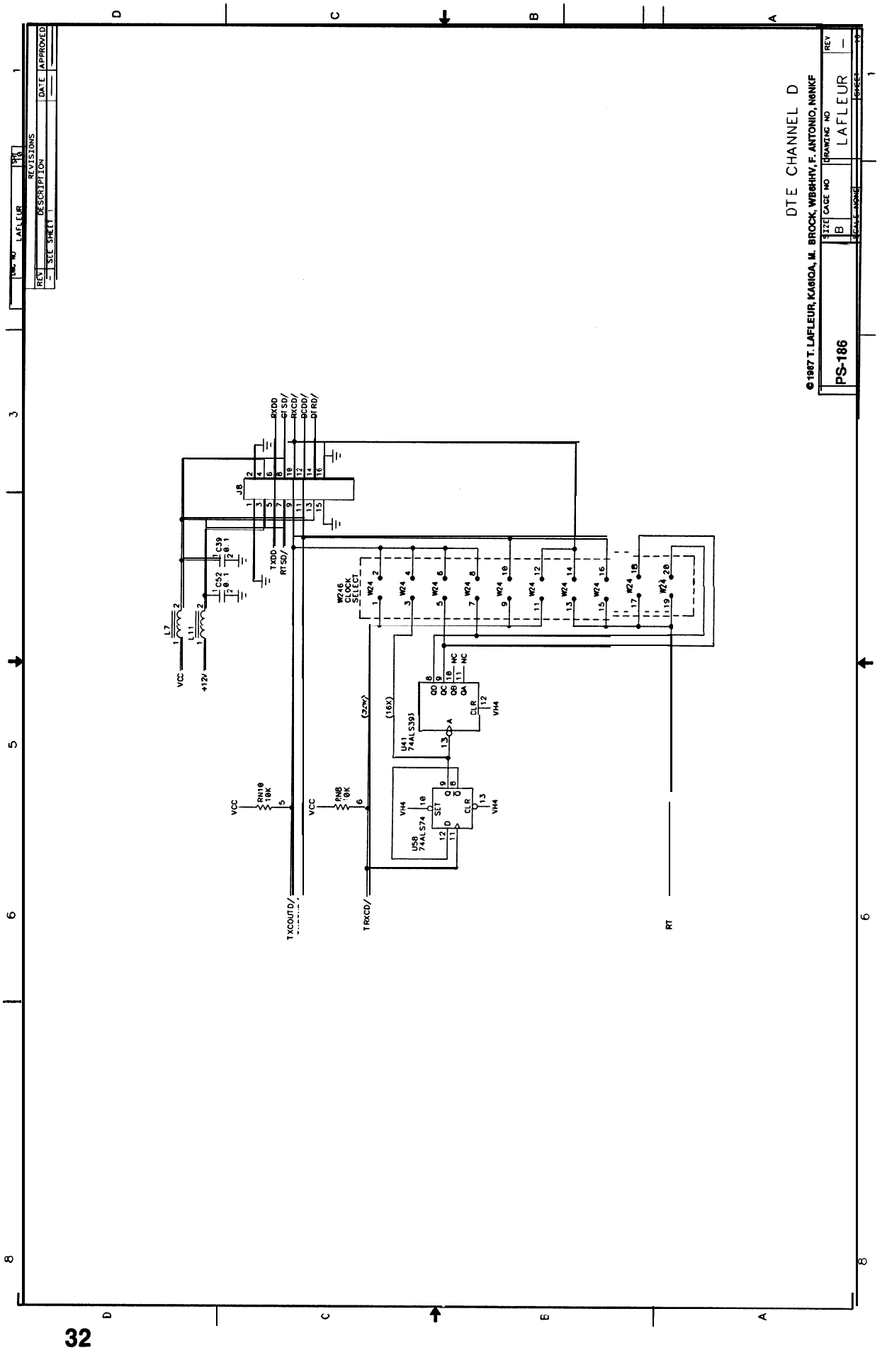
DTE CHANNEL B

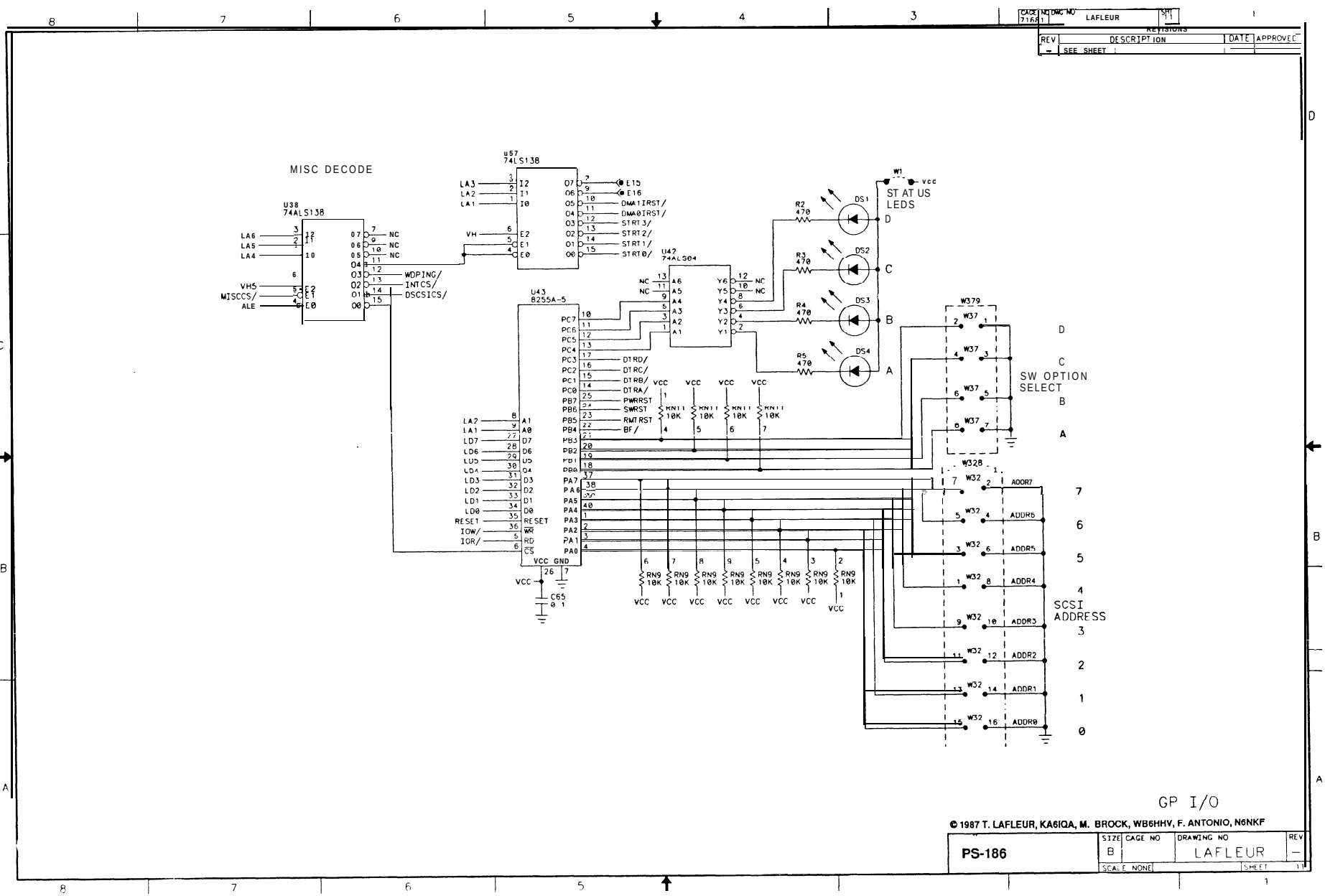
	SIZE	CAGE NO	DRAWING NO	REV
PS-186	B		LAFLEUR	-
	SCALE: NONE		SHEET	8

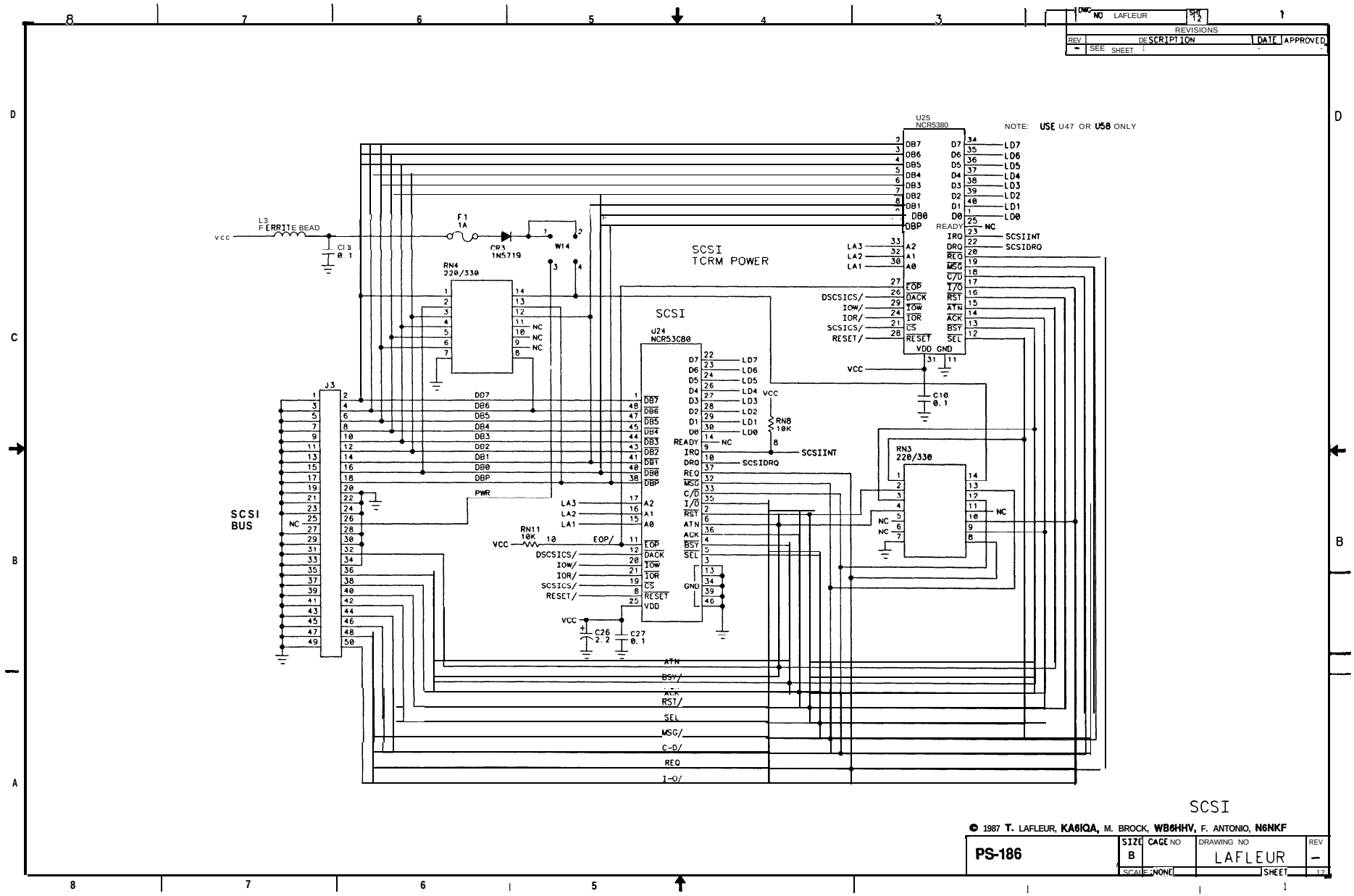


DTE CHANNEL C

PS-186	SIZE B	CAGE NO	DRAWING NO LAFLEUR	REV –
	SCALE NONE		SHEET 9	


DTE CHANNEL D
PS-186
LAFLEUR
© 1987 T. LAFLEUR, KA9IOA, M. BROCK, WB8HHV, F. ANTONIO, N6NKF
REVISIONS
DESCRIPTION
APPROVED
SEE SHEET 1
J8
RXDD
CTSD/
RXCD/
DCDD/
DTRD/
TXDD
RTSD/
C39
C52
L7
L11
VCC
+12V
W216
CLOCK
SELECT
U41
74ALS393
(32X)
(16X)
U58
74ALS74
RN10
10K
RN8
10K
TXCOUTD/
TRXCD/
RT

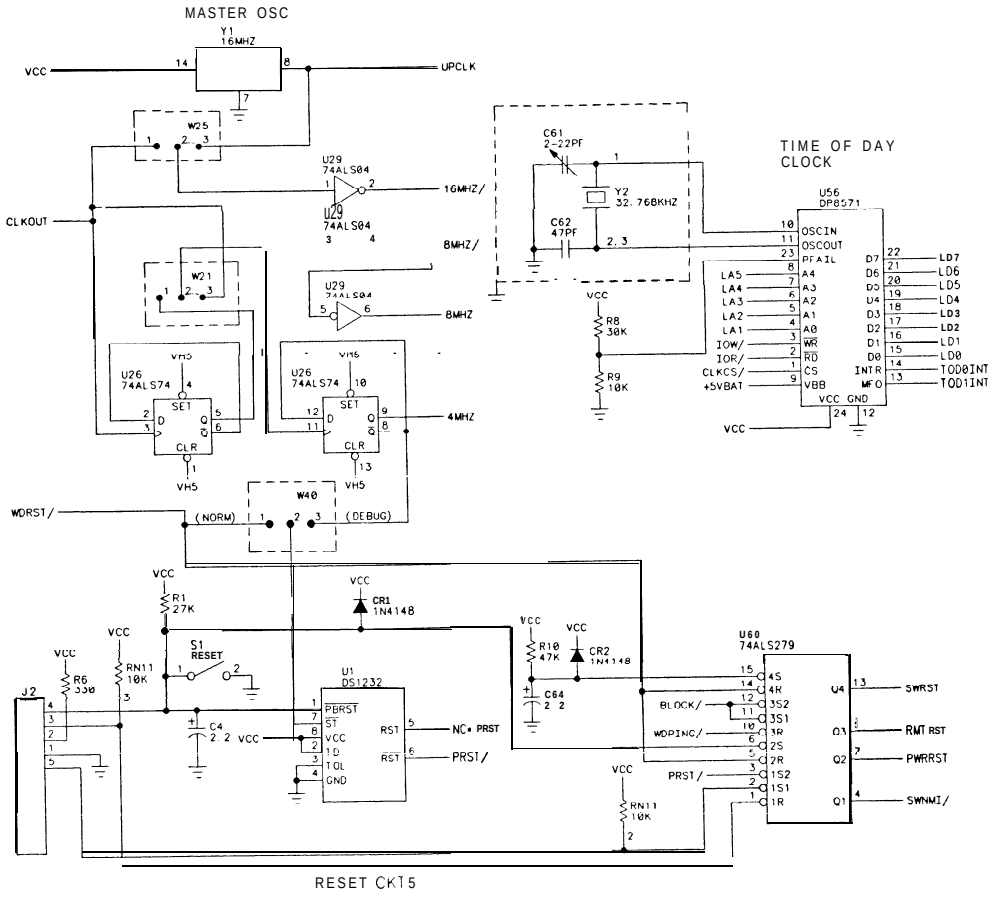

MISC DECODE
U38
74ALS138
U57
74LS138
U42
74ALS04
U43
8255A-5
W1
VCC
ST AT US
LEDS
DS1
DS2
DS3
DS4
R2 470
R3 470
R4 470
R5 470
D
C
B
A
W379
W37
SW OPTION
SELECT
W328
W32
SCSI
ADDRESS
7
6
5
4
3
2
1
0
ADDR7
ADDR6
ADDR5
ADDR4
ADDR3
ADDR2
ADDR1
ADDR0
LA6
LA5
LA4
VH5
MISCCS/
ALE
LA3
LA2
LA1
VH
WDPING/
INTCS/
OSCSICS/
DMA1IRST/
DMA0IRST/
STRT3/
STRT2/
STRT1/
STRT0/
E15
E16
DTRD/
DTRC/
DTRB/
DTRA/
PWRRST
SWRST
RWTRST
BF/
LA2
LA1
LD7
LD6
LD5
LD4
LD3
LD2
LD1
LD0
RESET
IOW/
IOR/
VCC GND
C65
0.1
KN11
10K
RN9
10K
GP I/O
© 1987 T. LAFLEUR, KA6IQA, M. BROCK, WB6HHV, F. ANTONIO, N6NKF
PS-186
SIZE B
CAGE NO
DRAWING NO LAFLEUR
REV –
SCALE NONE
SHEET 1/1
REVISIONS
REV DESCRIPTION DATE APPROVED
– SEE SHEET 1



PS-186

SCSI

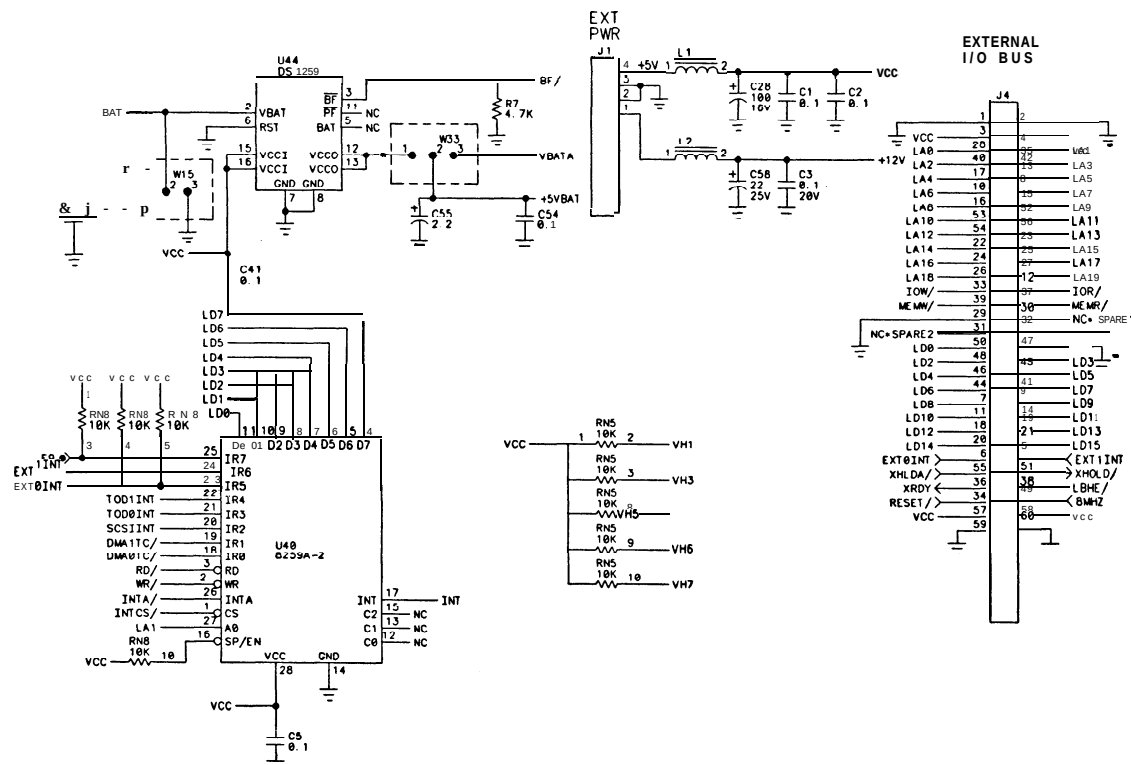
LAFLEUR



CLOCK AND RESET

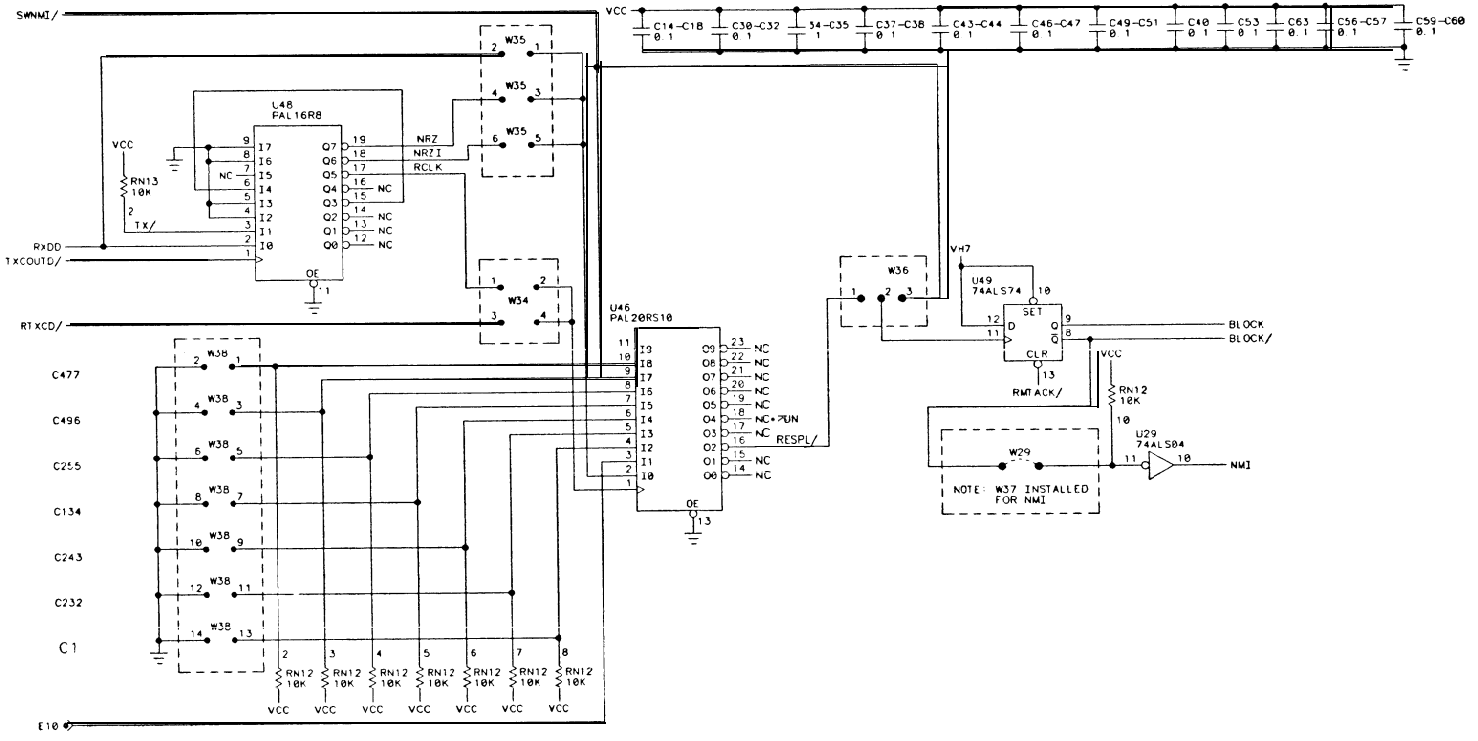
PS-186

SIZE B | CAGE NO | DRAWING NO LAFLEUR | REV 1
SCALE NONE | SHEET 13



INTERRUPTS AND EXT I/O

PS-186	SIZE B	CAGE No	DRAWING NO LAFLEUR	REV -
	SCALE NONE		SHEET 14	



REMOTE RESET

PS-186

The KISS TNC: A simple Host-to-TNC communications protocol

Mike Chepponis, K3MC

Phil Karn, KA9Q

ABSTRACT

The KISS[1] TNC provides direct computer to TNC communication using a simple protocol described here. Many TNCs now implement it, including the TAPR TNC-1 and TNC-2 (and their clones), the venerable VADCG TNC, the AEA PK-232/PK-87 and all TNCs in the Kantronics line. KISS has quickly become the protocol of choice for TCP/IP operation and multi-connect BBS software.

1. Introduction

Standard TNC software was written with human users in mind; unfortunately, commands and responses well suited for human use are ill-adapted for host computer use, and vice versa. This is especially true for multi-user servers such as bulletin boards which must multiplex data from several network connections across a single host/TNC link. In addition, experimentation with new link level protocols is greatly hampered because there may very well be no way at all to generate or receive frames in the desired format without reprogramming the TNC.

The KISS TNC solves these problems by eliminating as much as possible from the TNC software, giving the attached host complete control over and access to the contents of the HDLC frames transmitted and received over the air. This is central to the KISS philosophy: the host software should have control over all TNC functions at the lowest possible level.

The AX.25 protocol is removed entirely from the TNC, as are all command interpreters and the like. The TNC simply converts between synchronous HDLC, spoken on the full- or half-duplex radio channel, and a special asynchronous, full duplex frame format spoken on the host/TNC link. Every frame received on the HDLC link is passed intact to the host once it has been translated to the asynchronous format; likewise, asynchronous frames from the host are transmitted on the radio channel once they have been converted to HDLC format.

Of course, this means that the bulk of AX.25 (or another protocol) must now be implemented on the host system. This is acceptable, however, considering the greatly increased flexibility and reduced overall complexity that comes from allowing the protocol to reside on the same machine with the applications to which it is closely coupled.

It should be stressed that the KISS TNC is intended only as a stopgap. Ideally, host computers would have HDLC interfaces of their own, making separate TNCs unnecessary. [1 5] Unfortunately, HDLC interfaces are rare, although they are starting to appear for the IBM PC. The KISS TNC therefore becomes the “next best thing” to a real HDLC interface, since the host computer only needs an ordinary asynchronous interface.

[1] “Keep It Simple, Stupid”

2. Asynchronous Frame Format

The "asynchronous packet protocol" spoken between the host and TNC is very simple, since its only function is to delimit frames. Each frame is both preceded and followed by a special FEND (Frame End) character, analogous to an HDLC flag. No CRC or checksum is provided. In addition, no RS-232C handshaking signals are employed.

The special characters are:

Abbreviation	Description	Hex value
FEND	Frame End	C0
FESC	Frame Escape	DB
TFEND	Transposed Frame End	DC
TFESC	Transposed Frame Escape	DD

The reason for both preceding and ending frames with FENDs is to improve performance when there is noise on the asynch line. The FEND at the beginning of a frame serves to "flush out" any accumulated garbage into a separate frame (which will be discarded by the upper layer protocol) instead of sticking it on the front of an otherwise good frame. As with back-to-back flags in HDLC, two FEND characters in a row should not be interpreted as delimiting an empty frame.

3. Transparency

Frames are sent in 8-bit binary; the asynchronous link is set to 8 data bits, 1 stop bit, and no parity. If a FEND ever appears in the data, it is translated into the two byte sequence FESC TFEND (Frame Escape, Transposed Frame End). Likewise, if the FESC character ever appears in the user data, it is replaced with the two character sequence FESC TFESC (Frame Escape, Transposed Frame Escape).

As characters arrive at the receiver, they are appended to a buffer containing the current frame. Receiving a FEND marks the end of the current frame. Receipt of a FESC puts the receiver into "escaped mode", causing the receiver to translate a following TFESC or TFEND back to FESC or FEND, respectively, before adding it to the receive buffer and leaving escaped mode. Receipt of any character other than TFESC or TFEND while in escaped mode is an error; no action is taken and frame assembly continues. A TFEND or TESC received while not in escaped mode is treated as an ordinary data character.

This procedure may seem somewhat complicated, but it is easy to implement and recovers quickly from errors. In particular, the FEND character is never sent over the channel except as an actual end-of-frame indication. This ensures that any intact frame (properly delimited by FEND characters) will always be received properly regardless of the starting state of the receiver or corruption of the preceding frame.

This asynchronous framing protocol is identical to "SLIP" (Serial Line IP), a popular method for sending ARPA IP datagrams across asynchronous links. It could also form the basis of an asynchronous amateur packet radio link protocol that avoids the complexity of HDLC on slow speed channels.

4. Control of the KISS TNC

Each asynchronous data frame sent to the TNC is converted back into "pure" form and queued for transmission as a separate HDLC frame. Although removing the human interface and the AX.25 protocol from the TNC makes most existing TNC commands unnecessary (i.e., they become host functions), the TNC is still responsible for keying the transmitter's PTT line and deferring to other activity on the radio channel. It is therefore necessary to allow the host to control a few TNC parameters, namely the transmitter keyup delay, the transmitter persistence variables and any special hardware that a particular TNC may have.

To distinguish between command and data frames on the host/TNC link, the first byte of each asynchronous frame between host and TNC is a "type" indicator. This type indicator byte is broken into two 4-bit nibbles so that the low-order nibble indicates the command number (given in the table below) and the high-order nibble indicates the port number for that particular command. In systems with only one HDLC port, it is by definition Port 0. In multi-port TNCs, the upper 4 bits of the type indicator byte can specify one of up to sixteen ports. The following commands are defined in frames to the TNC (the "Command" field is in hexadecimal):

Command	Function	Comments
0	Data frame	The rest of the frame is data to be sent on the HDLC channel.
1	TXDELAY	The next byte is the transmitter keyup delay in 10 ms units. The default start-up value is 50 (i.e., 500 ms).
2	P	The next byte is the persistence parameter, p, scaled to the range 0 - 255 with the following formula: $P = p * 256 - 1$ The default value is P = 63 (i.e., p = 0.25).
3	SlotTime	The next byte is the slot interval in 10 ms units. The default is 10 (i.e., 100ms).
4	TXtail	The next byte is the time to hold up the TX after the FCS has been sent, in 10 ms units. This command is obsolete, and is included here only for compatibility with some existing implementations.
5	FullDuplex	The next byte is 0 for half duplex, nonzero for full duplex. The default is 0 (i.e., half duplex).
6	SetHardware	Specific for each TNC. In the TNC-1, this command sets the modem speed. Other implementations may use this command for other hardware-specific functions.
FF	Return	Exit KISS and return control to a higher-level program. This is useful only when KISS is incorporated into the TNC along with other applications.

The following types are defined in frames to the host:

Type	Function	Comments
0	Data frame	Rest of frame is data from the HDLC channel

No other types are defined; in particular, there is no provision for acknowledging data or command frames sent to the TNC. KISS implementations must ignore any unsupported command types. All KISS implementations must implement commands 0,1,2,3 and 5; the others are optional.

5. Buffer and Packet Size Limits

One of the things that makes the KISS TNC simple is the deliberate lack of TNC/host flow control. The host computers run a higher level protocol (typically TCP, but AX.25 in the connected mode also qualifies) that handles flow control on an end-to-end basis. Ideally, the TNC would always have more buffer memory than the sum of all the flow control windows of all of the logical connections using it at that moment. This would allow for the worst case (i.e., all users sending simultaneously). In practice, however, many (if not most) user connections are idle for long periods of time, so buffer memory may be safely "overbooked". When the occasional "bump" occurs, the TNC must drop the packet

gracefully, i.e., ignore it without crashing or losing packets already queued. The higher level protocol is expected to recover by "backing off" and retransmitting the packet at a later time, just as it does whenever a packet is lost in the network for any other reason. As long as this occurs infrequently, the performance degradation is slight; therefore the TNC should provide as much packet buffering as possible, limited only by available RAM.

Individual packets at least 1024 bytes long should be allowed. As with buffer queues, it is recommended that no artificial limits be placed on packet size. For example, the K3MC code running on a TNC-2 with 32K of RAM can send and receive 30K byte packets, although this is admittedly rather extreme. Large packets reduce protocol overhead on good channels. They are essential for good performance when operating on high speed modems such as the new WA4DSY 56 kbps design.

6. Persistence

The P and SlotTime parameters are used to implement true p-persistent CSMA. This works as follows:

Whenever the host queues data for transmission, the TNC begins monitoring the carrier detect signal from the modem. It waits indefinitely for this signal to go inactive. When the channel clears, the TNC generates a random number between 0 and 1.[2] If this number is less than or equal to the parameter p, the TNC keys the transmitter, waits .01 * TXDELAY seconds, and transmits all queued frames. The TNC then unkeys the transmitter and goes back to the idle state. If the random number is greater than p, the TNC delays .01 * SlotTime seconds and repeats the procedure beginning with the sampling of the carrier detect signal. (If the carrier detect signal has gone active in the meantime, the TNC again waits for it to clear before continuing). Note that $p=1$ means "transmit as soon as the channel clears"; in this case the p-persistence algorithm degenerates into the 1-persistent CSMA generally used by conventional AX.25 TNCs.

p-persistence causes the TNC to wait for an exponentially-distributed random interval after sensing that the channel has gone clear before attempting to transmit. With proper tuning of the parameters p and SlotTime, several stations with traffic to send are much less likely to collide with each other when they all see the channel go clear. One transmits first and the others see it in time to prevent a collision, and the channel remains stable under heavy load. See references [1] through [13] for details.

We believe that optimum p and SlotTime values could be computed automatically. This could be done by noting the channel occupancy and the length of the frames on the channel. We are proceeding with a simulation of the p-persistence algorithm described here that we hope will allow us to construct an automatic algorithm for p and SlotTime selection.

We added p-persistence to the KISS TNC because it was a convenient opportunity to do so. However, it is not inherently associated with KISS nor with new protocols such as TCP/IP. Rather, persistence is a *channel access* protocol that can yield dramatic performance improvements regardless of the higher level protocol in use; we urge it be added to *every* TNC, whether or not it supports KISS.

7. Implementation History

The original idea for a simplified host/TNC protocol is due to Brian Lloyd, WB6RQN. Phil Karn, KA9Q, organized the specification and submitted an initial version on 6 August 1986. As of this writing, the following KISS TNC implementations exist:

---

[2] To conform to the literature, here p takes on values between 0 to 1. However, fractions are difficult to use in a fixed point microprocessor so the KISS TNC actually works with P values that are rescaled to the range 0 to 255. To avoid confusion, we will use lower-case p to mean the former (0-1) and upper-case P whenever we mean the latter (0-255).

TNC type	Author	Comments
TAPR TNC-1 & clones	Marc Kaufman, WB6ECE	Both download and dedicated ROM versions.
TAPR TNC-2 & clones	Mike Chepponis, K3MC	First implementation, most widely used. Exists in both downloadable and dedicated ROM versions.
VADCG TNC & Ashby TNC	Mike Bruski, AJ9X	Dedicated ROM.
AEA PK-232 & PK-87	Steve Stuart, N6IA	Integrated into standard AEA firmware as of 21 January 1987. The special commands "KISS ON" and "KISS OFF" (!) control entry into KISS mode.
Kantronics	Mike Huslig	Integrated into standard Kantronics firmware as of July 1987.

The AEA and Kantronics implementations are noteworthy in that the KISS functions were written by those vendors and integrated into their standard TNC firmware. Their TNCs can operate in either KISS or regular AX.25 mode without ROM changes. The TNC-1 and TNC-2 KISS versions were written by different authors than the original AX.25 firmware. Because of the specialized development environment used for the TNC-1 code, and because original source code for the TNC-2 was not made available, the KISS authors wrote their code independently of the standard AX.25 firmware. Therefore these TNCs require the installation of nonstandard ROMs. Two ROMs are available for the TNC-2. One contains "dedicated" KISS TNC code; the TNC operates only in the KISS mode. The "download" version contains standard N2WX firmware with a bootstrap loader overlay. When the TNC is turned on or reset, it executes the loader. The loader will accept a memory image in Intel Hex format, or it can be told to execute the standard N2WX firmware through the "H"[3] command. The download version is handy for occasional KISS operation, while the dedicated version is much more convenient for full-time or demo KISS operation.

The code for the TNC-1 is also available in both download and dedicated versions. However, at present the download ROM contains only a bootstrap; the original ROMs must be put back in to run the original TNC software.

8. Credits

The combined "Howie + downloader" ROM for the TNC-2 was contributed by WA7MXZ. This document was expertly typeset by Bob Hoffman, N3CVL.

9. Bibliography

1. Tanenbaum, Andrew S., "Computer Networks" pp. 288-292. Prentice-Hall 1981.

2. Tobagi, F. A.: "Random Access Techniques for Data Transmission over Packet Switched Radio Networks," Ph.D. thesis, Computer Science Department, UCLA, 1974.

3. Kleinrock, L., and Tobagi, F.: "Random Access Techniques for Data Transmission over Packet-Switched Radio Channels," *Proc.* NCC, pp. 187-201, 1975.

4. Tobagi, F. A., Gerla, M., Peebles, R.W., and Manning, E.G.: "Modeling and Measurement Techniques in Packet Communications Networks," *Proc. IEEE,* vol. 66, pp. 1423-1447, Nov. 1978.

5. Lam, S. S.: "Packet Switching in a Multiaccess Broadcast Channel", Ph.D. thesis, Computer Science Department, UCLA, 1974.

6. Lam, S. S., and Kleinrock, L.: "Packet Switching in a Multiaccess Broadcast Channel: Dynamic Control Procedures," *IEEE Trans. Commun.,* vol COM-23, pp. 891-904, Sept. 1975.

[3] For "Howie", of course.

7. Lam, S. S.: "A Carrier Sense Multiple Access Protocol for Local Networks," *Comput. Networks*, vol 4, pp. 21-32, Feb. 1980

8. Tobagi, F. A.: "Multiaccess Protocols in Packet Communications Systems," *IEEE Trans. Commun.*, vol COM-28, pp. 468-488, April 1980c.

9. Bertsekas, D., and Gallager, R.: "Data Networks", pp. 274-282 Prentice-Hall 1987.

10. Kahn, R. E., Gronemeyer, S. A., Burchfiel, J., and Kungelman, R. C. "Advances in Packet Radio Technology," *Proc. IEEE* pp. 1468-1496. 1978.

11. Takagi, H.: "Analysis of Polling Systems," Cambridge, MA MIT Press 1986.

12. Tobagi, F. A., and Kleinrock, L. "Packet Switching in Radio Channels: Part II - The Hidden Terminal Problem in CSMA and Busy-Tone Solution," *IEEE Trans. Commun.* COM-23 pp. 1417-1433. 1975.

13. Rivest, R. L.: "Network Control by Bayessian Broadcast," Report MIT/LCS/TM-285. Cambridge, MA. MIT, Laboratory for Computer Science. 1985.

14. Karn, P. and Lloyd, B.: "Link Level Protocols Revisited," *ARRL Amateur Radio fifth Computer Networking Conference*, pp. 5.25-5.37, Orlando, 9 March 1986.

15. Karn, P., "Why Do We Even Need TNCs Anyway," *Gateway*, vol. 3 no. 2, September 5, 1986.

DIGITAL SIGNAL PROCESSING
and Amateur Radio
by Dr. Thomas A. Clark W3IWI and Dr. Bob W. McGwier N4HY

Tom calls the necessary ingredient magic, and Paul Rinaldo, W4RI, calls the necessary ingredient an incubator[1] [2]. Whatever you choose to call it, we have lacked the necessary ingredient in ham radio to broadly apply the collection of techniques known as *digital* signal *pro*cessing (DSP). The magic in the amateur satellite service has been provided by AMSAT[3]. In the digital communications modes, the incubator was supplied by TAPR[4]. AMSAT and TAPR have joined to support an incubator project to achieve the broad use of DSP in amateur radio.

DSP is a field which has its roots in the mathematics of Newton, Gauss, and Fourier. These people lived hundreds of years ago. Why are we stessing these ancient techniques? Primarily because a much newer technological revolution has taken place which allows us to apply DSP techniques in increasingly complex combinations. Of course, we are talking about computers in general and microprocessors in particular. Until the very recent past, even microprocessor elements were primarily used in proprietary special purpose devices with specific tasks in mind. For example, there are DSP chip sets manufactured by several companies that are totally dedicated to being Bel-212 modems for phone line use.

We will not delve into the theory of Fourier analysis, linear systems, ARMA models, and finite impulse response filters here. However, a little study of these technical topics reveals that we need only do a handful of mathematical operations well. The primary ones are multiply, accumulate, and shift contiguous data locations in memory. If we were to design a processor that did these things well, and ignored a lot of the things general purpose microprocessors are supposed to do, we would open our systems to the power of these DSP techniques. Fortunately for us, we don't have to design these chips. Possibly the best known family of DSP chips today is the TMS320 family of DSP microprocessors (DSP engines). In fact, the three operations above can be combined into *one machine instruction* on the TMS320 family. It is what makes these chips one of the idealized sets for most DSP operations. There are others, including the Motorola DSP 56000 which have recently begun to flow to users of DSP engines. For others, and a general discussions of these chips see[5].

Why should all this excite us? Maybe a few examples are in order. A well known novel, which used DSP techniques to the advantage of submariners, was Tom Clancy's "The Hunt for Red October." The signal processing done in that book enabled the "hunt" to be sucessful. The story was fiction but the techniques on which the plot was based are very real. CAT (computer aided tomography) scans have greatly reduced the need for inherently dangerous exploratory surgery thanks to DSP. However, the funds for research and development have come primarily from the telecommunications industry. The ability to do very complex DSP operations on small devices has greatly reduced the capital costs of maintaining and growing the telecommunications that now surround the globe. The ISDN (integrated services digital network) will be a reality in the near future thanks in a large part to digital techniques.

BUT WAIT THERE'S MORE

What has this to do with amateur radio? Imagine that on Monday Bob wants to talk to Tom on 20 meters and wants to use ACSSB (amplitude compandored SSB) for a better signal to noise ratio. On Tuesday Tom finds that 20 meters is dead but needs to talk to Bob again, so he will use a voice encoding scheme and send the digitized voice over the packet network between them. But wait, there's more. On Wednesday, Bob wants to transfer a large file that resulted from the previous two days of conversation to Tom and wants to do it at 9600 BPS. But wait, these guys are really active, there's more. On Thursday, Tom wants to gather weather maps from the NOAA satellites. On Friday, Bob wants to talk to Tom but apparently transmitter is acting up and he needs an audio spectrum analyzer. But wait there is more to "DSP week." On Saturday, it's an EME weekend and Bob and Tom want to try getting each others echoes off the moon. Alas! Murphy has struck and both have lost their high power amplifiers. But wait there is more. On Saturday, they decide to try Fuji Oscar 12 PSK packet and its bulletin board in space. A few years ago, you would have said these guys are millionaires or loco since all this can't be done anyway with what we have. Each of these operations is characterized by one important feature

On a DSP chip **IT'S ONLY SOFTWARE.**

ACSSB is almost too easy on DSP engines. LPC-10 and ADPCM (linear predictive coding and adaptive pulse coded modulation) have been around for a while to do encoding of digitized voice. The PSK modems and the WEFAX-APT demodulation are both based on a phase locked loop (PLL) which is straightforward on a DSP chip. The spectrum analyzer and the EME weak signal detection technique both use the same DSP process (FFT) as their basis as decsribed below. In addition to these examples, one you can make modules for all the standard modems and some non-standard modems at least up to 2400 bps on the least expensive of these chips (TMS320C10). The heart of the 9600 bps demodulator can be done on this chip and the transmitter is trivial. On a single generation later

the least expensive chip (and still inexpensive) you can do the entire 9600 bps modem. When Texas Instruments finally releases the TMS320C30, you can do a 56 Kbps MSK modem with coherent demodulation (optimal) and have microprocessor power left over to do the entire AX25 protocol and HDLC. This chip, when coded carefully will sport 33 Megaflops (!!) and that is about 3/8-ths of the fastest code N4HY has ever been able to code on a Cray-1.

We have emphasized some fairly advanced applications. In fact, these chips lend themselves very well to the more mundane tasks of filters of many types: complex bandpass, lowpass, and highpass filters of many types are very straightforward on these chips. An optimal and adaptive woodpecker filter that can "tune" to different pecking rates is also achievable on this type of chip. The list of things you can do on these chips is enough to write an entire textbook on digital signal processing and there are books available on these topics[6].

A Little Magic

TO begin this project we are negotiating for 20-25 boards manufactured by Delanco-Spry in Silver Spring, Md. On them is the Texas Instruments TMS320C10-25. These boards fit into a standard long expansion slot on almost MS-DOS based machine (see figure 1). Our final BUT WAIT THERES MORE is that all the functions mentioned in the DSP week above can be done on these boards with high level support from the DOS machine. Some of the software exists and has been tested and is being improved. If it were all done there would be no need for a project and we would concentrate on getting wealthy. We have concentrated our initial efforts on the spectrum analyzer and weak signal work with a great deal of success. The spectrum analyzer is the functional equivalent (with a few less bells and whistles) of a $10000 machine manufactured by a reputable company here in the U.S. When most of us think of a spectrum analyzer, we envision a device which is a tunable receiver slaved to an oscilloscope. The frequency of the receiver is swept and the receiver's detector output is displayed. At any one instant, the receiver is tuned to only one frequency. If the receiver has a given detector bandwidth and we are using some total sweep width, then we can define a useful ratio N to be

N = sweep width / bandwidth

where N is typically 100-200 and is the number of independent frequency channels that are being scanned. On any one channel we spend only $1/Nth$ of the time and our ability to see a weak signal is severely comprimised. In the case of a spectrum analyzer done with DSP techniques, we can choose to have N independent detectors, each of which operates all the time. To do this? we have to take at least 2N data samples of the wideband input signal (corresponding to the sweep width) with a high-speed A/D converter, and then do mathematical magic on the 2N numbers; the magic most often applied is the Fast Fourier Transform (FFT). You make N be some power of 2 (like 128, 256, 512, 1024). With the DSP chips and boards we can now buy for less than the price of a weekend in Los Angeles or an 8877 on 2 meters, N=512 with a channel bandwidth of a few Hz and a total spectral bandwidth of a few kHz is feasible, See figure 2 for a pictorial representation of the two types of analyzers.

If you add up the cost of using special purpose hardware to do the other tasks in the DSP week, you can see that the savings are tremendous and indeed make some of these things possible where they were at best difficult before. The primary example is the EME without high power amplifiers. We hope this project will produce the other things in our DSP week. The structure of the project will follow a set of rough guidelines. We are looking **for** a few people who are capable of writing (or learning to write) TMS320 assembler to write the functional modules that will be used as building blocks. The applications in this project will be controlled in a high level language on the PC. We have tentatively settled on "C" as the natural system language for these applications. Already in use are Borland's TurboC^{tm} and Microsoft's MSC-5.0^{tm} and Aztec "C". We are looking for people who can program "C" to help produce these neat applications for the modules on the board. We also have a place for a few people who will be strictly "users" to critique the efforts of the people doing the coding. As applications become available, we will either publish them or make them available to the amateur community through the commercial vendors in those instances where the vendor either paid for the development by funding the work or acquires a license from the project participants. The commerical route is for those who have (say) a C-64 and want a widget that plugs into their computer and -provides these neat applications. In the case of ACSSB, DSP will provide an easy way to get pilot tone coded compression without! having to adapt someone else's reject boards or modifying your equipment. For packet radio, for the first time we will have adaptive equalization working for us. If you are interested in participating in this project, write or call us and we'll be glad to put you to work! We believe this project will provide exciting new capabilities

REAL SOON NOW!

[1] Thomas A. Clark, a talk given at the 1987 Trenton Computer Festival.

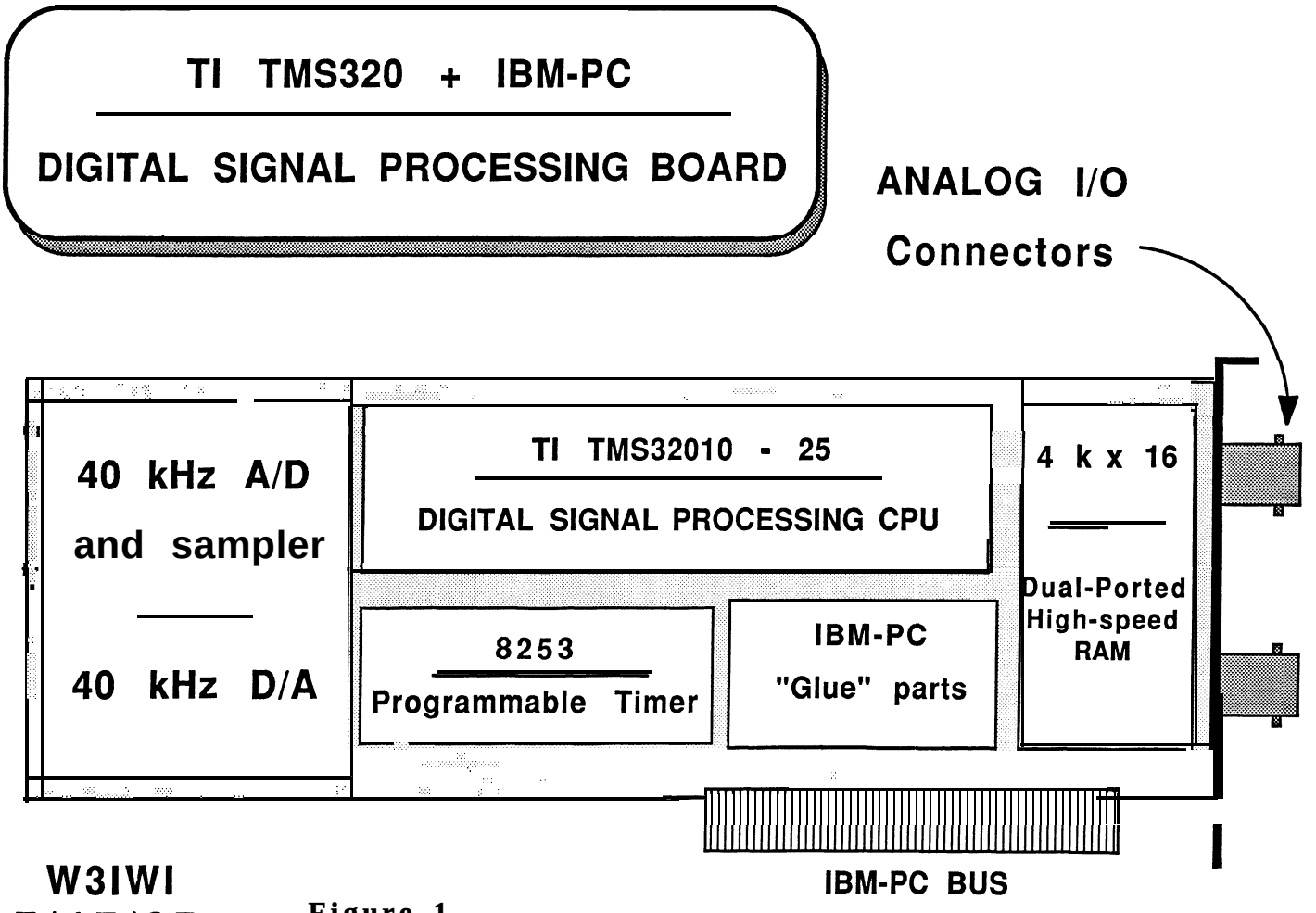
[2] Paul Rinaldo, W4RI, "Empirically Speaking", *QEX*, April 1987.

[3] In the U.S., AMSAT, Inc., P.O. Box 27, Washington, D.C. 20044

[4] TAPR, Inc., P.O. Box 22888, Tuscon, Arizona 85734..

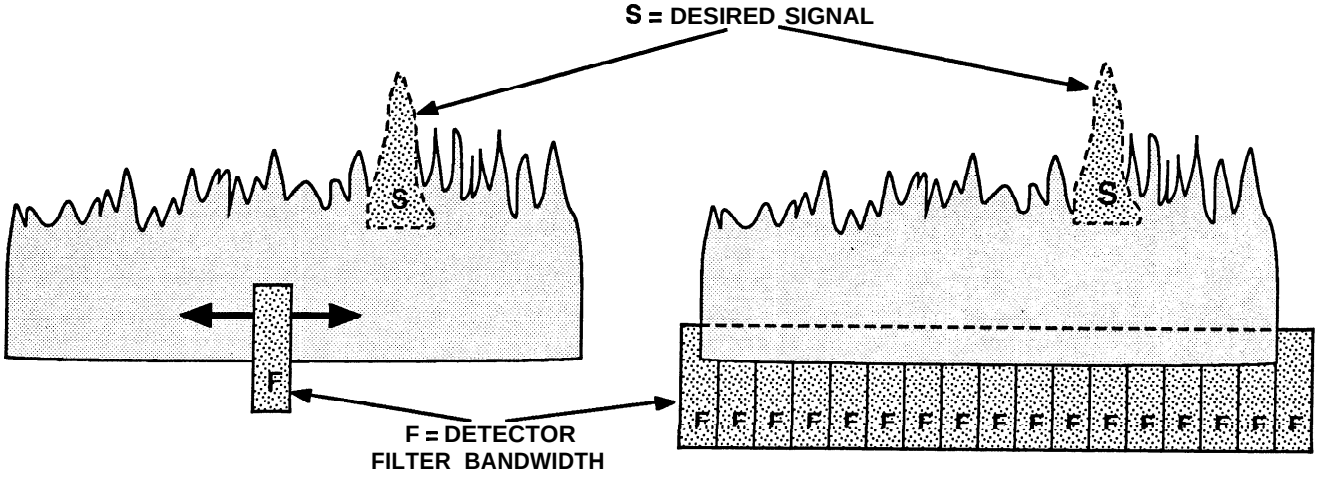
[5] Aliphas, Amnon and Feldman, Joel, "Digital Signal Processing Chips", *IEEE Spectrum*, June 1987, pp. 40-45

[6] Rabiner, R. and Gold, B., *Digital Signal Processing.* Prentice-Hall, Englewood Cliffs, N.J., 1975.



W3IWI
7/17/87

Figure 1



Swept Single Filter Spectrum Analysis Multi-channel Spectrum Analysis

Figure 2

A DUPLEX PACKET RADIO REPEATER APPROACH TO LAYER ONE EFFICIENCY

Robert Finch, N6CXB Scott Avent, N6BGW
c/o South Coast Radio Relay
4446 Rosebank Drive
La Canada, California 91011-1928

PURPOSE

In January, 1984, South Coast Radio Relay, a Southern California Amateur Radio group specifically chartered to experiment in state of the art message handling techniques, modified its two meter duplex voice repeater in Glendale, California, for full time digital communications. This operation has remained on the air since that time, under both the original callsign; N6TD/R and the since adopted call; N6GPP/R. This operation has undergone some specific changes in the past several years, but the basic user interface has essentially remained the same. This paper will examine the basic operational characteristics employed at the repeater site, specific to the repeater itself, and will aid the reader in duplicating the methods used to achieve the promise inherent in local area duplex packet operation.

THE PROMISE

Packet has been sold to the amateur community as an error free way of 'message handling'. That in and of itself, has been realized. However the initial means for a packet station to be used is through the simple LAN (local-area-network), made available to the user to actually accomplish any message transmission to the intended destination. The experimenters in our ranks are continuing to push for higher efficiency levels of inter-networking links to allow for more distant destinated traffic and higher volumes of in transit traffic. As a group, we quite often leave the 'end-user' ham to the low volume, low efficiency, simplex LAN. In some areas this LAN is on the same channel (frequency) as the high volume wide area networks. These wide area networks are double trouble for the local user, Not only is the traffic volume high, but it can also be automated and characteristically highly persistant in occuping the available time slots. To allow for continued interest and growth we have to change this, where possible. The authors hope to shed some light in this often overlooked area by presenting the method we have been using to attempt to accomplish some of these goals.

BACKGROUND

The current layer two protocol in use is dependent on the use of a CSMA-CA/CD (carrier sensed multiple access - collision avoidance/ collision detection) usage contention scheme to maximize channel efficiency in real-time. In order for this to work correctly it is assumed that every station in the network can correctly hear every other station within the network's basic transmission area. A group of stations in an ideal network topology can acheive this time sharing on a single frequency channel. In the real world this cannot be counted on. In practical terms it has shown itself not readily capable of this time and time again. As we shall touch upon later, this has proven to be a real thorn in the side of local amateur packet radio operations.

A paper has been previously presented at the fourth ARRL networking conference examining the relative merits of various layer one approaches, and makes excellent background reading. In this paper; PACKET RADIO TIMING CONSIDERATIONS, the author (David Engle, KE6ZE,) presented AX.25 specific timing information, and examined the relative channel through-put efficiency of each of three 1200 baud AFSK approaches. Of course the most efficient was the direct connect, i.e., without any form of repeating path involved. The second most efficient was the duplex repeater (in their experiments a regular voice repeater with a 500ms transmitter key-up delay). The final option examined, and the least efficient was the single frequency repeater or 'digipeater'. At the end of this earlier paper the author made several suggestions on improving the duplex repeater performance based on the experimentation he made, attempting to approach the measured efficiency of a 'direct-connection' simplex layer-one connection. There are differences in the specific approaches taken between our implementation and the previous author's recommended duplex repeater configuration. We do not always agree on specfic design 'practices', but our current repeater has exceeded our original design goals and in operation can essentially equal the efficiency of 'direct-connection simplex packet. Additionally we significantly increase the coverage area of the average user's station within his local area network.

OUR SOLUTION

The basic block diagram of N6GPP/R repeater, in 'black-box' form, would appear much like a regular voice repeater with the addition of a RS-232 port attached to allow for 'inter-network' additions. In fact this inter-netwroking was accomplished some years ago with the addition of the N6GPP-1 145.01 mhz. network access computer and a Motorola Mocom mobile radio. This computer is basically a Xerox 820 modifed to KE3Z specifications and running a modified version of his ARRL control firmware. The repeater itself is a late model commercial Motorola Micor duplex repeater. The principle advantage to this model repeater is its fully solid state, crystal controlled operation, without the use of any 'switching' relays, in the the control of its basic operation(s). This allowed us to cut the packet recognition and

repeater transmitter turn on time to less than 30 ms.

The basic transmitter multiplier and modulation circuits are on continuously and only the solid state P.A. is enabled after valid packet recognition has taken place. In operation the R.F. transmitter 'turn-on time' is well under 100 micro secs., the balance of the 30 ms. being the reception circuitry. Additionally, within the transmit audio path is a custom 'zero-voltage' fet switch controlled configuration which allows us to maintain a tone free squelch tail. We also eliminated the stock limiting and clipping stages. This reduction of audio processing circuitry allows us to realize a perfectly level controlled sine wave with essentially no distortion due to any 'audio-processing' and therefore no splatter filtering is needed. The audio can be fully modulated to desired levels without worry of any undesirable or otherwise uncontrollable products in the R.F. output.

Working backwards from the transmitter audio circuits is the modulating 'modem'. In our case there is only one single modem in use for the entire demodulation, modulation and the outside world RS-232 connection. However the basic design is much more easily thought of, initially, as two modems connected back to back, so to speak. The modulating modem must be free of any distortion as previously mentioned, and some scheme for assurance that no start-up artifacts occur during the 'settling' of the audio tone during turn-on is required. To accomplish this a modem assuring zero-crossing tone switching is employed. In our case we settled on using a digitally based commercial 202 modem, which could be confiqured as separate receive and transmit sections and still allow for some intergration of hardware control signalling. The modem is set-up for 20ms total time for acquisition and 8.5ms RTS (request to send) to CTS (clear to send) turn around time. This turn around time is utilized for two basic functions. First the afforementioned settling of the modem generated audio is assured. Secondly the incoming RTS can be controlled on a first in-first out basis with the outside network and the duplex receive channel contending for control.

This high-speed 'turn-around' of request to send to clearance for transmission is the key to keeping the efficiency of channel through-put high enough to realize our goal of essentially transparent operation, while still allowing for the addition of various incoming sources (outside networks). By using hardware control via the RS-232 standard; full contention resolution can be accomplished with the addition of this simple first in first out circuitry. This still allows efficiency to remain at a high level.

By using a standard six-wire RS-232 implementation (ground, TXData, RXData, request to send, clear to send, and data carrier detect) multiple source and destination connections can be implemented via simple fan-out of detection signals to various contention control devices. Specifically the digital demodulation of received information generates a DCD (data carrier detect) signal which is fanned out to the first-in first out contention circuit and the input of the Xerox 820 inter-network routing computer. The output of the same computer is also routed to the first-in first-out circuit, which in turn generates the RTS signal to the audio modulation (portion of the) modem. Similarily the first-in first-out circuit handles the routing of RXD (RXData) and TXD (TXData) and the CTS (clear to send) signal which will allow inhibition of the Xerox's sending circuitry if the computer is not actually rquesting transmission first. After this circuit desides who has control of the repeater transmitter it then goes on to the finally key-up the transmitter and gate the audio thru to the modulator. Standardization on this RS-232 hardware control allows us to mix the non-intelligent nature of the duplex in to out signalling with the computer based routing of the xerox computer. In fact the new NETROM (tm) and other intelligent controllers are offering just this specific RS-232 standard hardware control, which allows us freedom for future changes and enhancements of wide-area-network integration within the current system (N6GPP/R along with N6GPP-1) to enhance the inter-networking of our local area network.

The receiver portion of the repeater is also very stripped down. Packet recognition is based around two distinct circuits. These circuits combine to acheive fast recognition without needless 'falsing' out the repeater output frequency due to 'spurious' and otherwise unneccessary signals at the receiver's R.F. input. The first is the use of the demodulation section of the above mentioned 202 modem for not only conversion of audio to digital signaling but also to allow the use of the inherent tone recognition circuitry within the standard commercial 202 modem. This timing is set to approximately 20 ms as a good compromise without being too long (slow). We found while this was sufficiently fast, when coupled with the transmitter timing, it was not totally sufficient in eliminating the falsing. To augment this required some additional circuitry. This circuitry is not neccessary in all implementations and some explanation is in order.

While we feel that the turnaround and recognition circuitry within a standard commercial 202 is ideal to meeting our goals as mentioned above, restraints can raise their ugly head(s) and did in our case. In a crowded R.F. spectrum as is found in a large metropolitan area, such as we are priviledged to have here in greater Los Angeles, the compromise afforded had to be enhanced by the use of a specific 'packet-recognition' circuit. This simple circuit essentially discards a packet when the repeater is initially started in order to allow verification of a large number of transitions within a short time interval. The assumption is that the demodulator with-in this time interval will not transition its RXData out stream a large number of times within a small timing window unless the audio signal is a valid 1200 baud data stream. The window is formed by the same DCD (data carrier detect) output signal that forms the basis for the input to the above mentioned first-in first-out contention circuitry and is then used to request the data transmission repeat. Making this recognition circuit re-triggerable and enabling the DCD connection to the contention circuitry as the result of valid packets, verification is a definite compromise. However with the interval for restart of this circuit set at 2 mins. 12 secs. we have found as a

practical matter that it is used only to initally enable the repeater as intended even with the generally slow packeteer-typist. Making it a retriggable circuit allows arthritic typists access too.

Of minor significance is the repeater's T.O.T. (time-out-timer). All digital repeaters should employ these, as is standard practice with the garden variety voice repeater. As you can see by now, our repeater doesn't need to run with the more commonly found repeater control shelf. In fact a digital repeater's control circuitry can be easier to construct, For completeness we would like to point out we have a time out timer.

Of considerably more significance is the phase-coherent R.F. discriminator (detector) and the squelch circuitry found in our Micor receiver. The modem is directly connected to the discriminator. This direct connection mirrors the intent behind the essentially direct connection of the 202 modem's modulation output thru the fet switching to the R.F. modulator. The cleanest possible signal allows for the least amount of error rate increases due to distortions. The standard audio squelch is **not** needed in the conventional usage (but could be beneficial in reducing some remaining 'noise key-ups'), but since it is part of distortion producing audio processing cicuitry it isn't used. As mentioned above, the design goal was to keep the entire audio chain as clean as possible. We are currently investigating possible inclusion of a suitable squelch circuit not requiring any in line processing circuitry. While the use of product detector is not ruled out in the adopted layer **one** standard for two meters (1200 baud AFSK), it is a neccessary requirement to use a phase-coherent detector for any duplex digital repeater based on a wide bandwidth, true FSK, layer-one. In our case, the phase coherent detector in use is supplied with the standard Micor package we used.

In passing we'd like to mention that the control system does not maintain any on-site ID'ers. We handle this function as a beacon signal generated by one of the channel hosts on a regular basis, timed to assure that we are in compliance with the applicable regulations on repeater identification,

The use of a hang-time (squelch tail) has received much debate as used in our operation, and the various advantages and disadvantages are still discussed with some regularity after three-plus years of operation. Initially we found that 2 meter voice users couldn't tell that there was a repeater on our frequency pair without it. This was especially troublesome for both digital and voice users back then. While perhaps not a problem these days, by making the 'tail' longer than **one** retry interval, we incured essentially no problems with its inclusion. This helped back in the days when TAPR was shipping serial number 350 of the original tnc-1, and packets sounded like 'intermod' to the rest of the amateur **community.** By creating a tail after every series of packets sent, while the average ham might not understand what the signal is for, he at least knew there was something resembling a repeater on the frequency. With the addition of the newer version two of the layer two protocol to the scene, there has been some mention of changing this 'hang time' to longer than one 'response' interval. However we have found that in most all cases, setting user's response time to zero, and leaving the hang time as is, an even better solution. This allows us to accomodate all versions of user's operating firmware. The channel has such a good balance between receiver and transmitter coverage areas that any one packet user is virtually assured that his packet will be heard by all users, and we might mention that the older version one (layer two,) protocol runs especially well on duplex packet networks based on our experience.

The above mentioned balance to coverage areas of the repeater's receiver and transmitter, in practical terms, is more important than in a standard voice repeater. In fact the transmitter's coverage should be somewhat larger than the receiver%. To assure that the 'hidden-terminal' problems of simplex packet digipeaters aren't unneccessarily duplicated this balance becomes critical. When your packets are heard by all other users the number of users can be maintained at elevated levels compared to the average simplex repeater channel, which by its naturally occuring coverage topology cannot assure against easily resolvable collision avoidance and detection. (This AX.25 characteristic has been enumerated, discussed, and generally beated about in so many places and publications in the past several years, along with proposed modifications to the protocol etc. to help minimize its effects. We do not want to go into a longish discussion here. Suffice it to say that the layer two protocol, especially the original version one, is in light of this; a much better duplex protocol than simplex. In fact the apparent real-time loading fiqures, i.e. the level of absolute time in use for packet transmissions until the channel clogs up and is unusable for a time, is significantly higher in duplex compared to simplex operation. And the 'loading' curve that preceeds this clogging, in terms of the maximum number of pending packets serviced, is apparently much less steep with duplex packet.)

BACK TO THE USER

This discussion would not be complete without bringing up the enhanced operation the users enjoy. Beyond the higher efficiency local area network operation we can enjoy the novelty of true roundtable operation. By utilizing the essentially true to form CSMA-CA/CD nature of the layer two protocol that duplex affords, we connect up in pairs and hold discussions while in full channel monitoring mode with large numbers of 'conversational-mode' packet users. This is an excellent way to operationally verify the correct repeater transmission to reception balance. (As well as any one user's similar balance.) When a single user can request the retry of a 'lost' packet to the sender for the entire local area network, then excellent free-flow of discussion can be realized with a minimum of lost time in distributing 'information* within the network.

Another and perhaps the most important advantage to duplex's efficient use of the uniquely radio specific portion's of the layer two protocol in use is the variable 'DWAIT' and 'FRACK' timing intervals. By encouraging users to remain at minimum neccessary values, the non-real time 'channel-hosts' (in our case the N6BGW-9 auto-

forwarding mailbox and the N6CXB-1 database and file-server,) can use elevated values. This assumes that everyday users are at a low-volume usage per each station and therefore are assured transmission priority over the the higher volume per station 24 hour hosts. The hosts will wait for a longer free-time interval before transmission of pending packet traffic. By also keeping the host's packet sizes high and custom tailoring the FRACK intervals among the hosts, the hosts are not known as the channel hogs that some of the more popular simplex packet hosts are.

IN CLOSING

The authors are hopeful that this paper and our 'local area network' implementation will encourage other groups to help out their so often forgotten 'user communities', by taking some of the ideas mentioned here and using them to enhance the neighborhoods they live in.

PACKET RADIO DEVELOPMENTS OVER THE LAST YEAR

Terry Fox, WB4JFI
Vice-President, AMRAD
7877 Hampton Village Pass
Annandale, VA 22003

ABSTRACT

Over the last year, AMRAD has been continuing it's work on the development of packet switch hardware and software. Our progress has been slower than we had hoped, due to our interests in other projects, such as spread spectrum. The hardware portion of our packet switch has made better progress than the software. Over the next year we hope to make more progress in software development. What follows is a brief description of what I think have been some significant advances over the last year, along with some of my current thinking in packet network development.

Packet RF Development

Since last year there has been some movement in packet RF hardware. No longer is 1200 bps the fastest speed generally available. Kantronics has been shipping their 2400 bps packet boards (KPC-2400) for a while now. While this system has met with somewhat mixed reviews by the packet community, it does represent an advance in RF technology.

AEA announced at Dayton their new 220 mHz 9600 bps RF Modem/Radio. This unit is designed to be used either as a voice radio (with included microphone) or as a 9600 bps RF modem for packet work. One nice result of this is that voice operation can be used to align the RF path and test the units out, then switched to data mode for normal operation. As of this writing, I am not aware of any units actually shipped by AEA, but I guess they will be shipping soon.

Another high-speed packet rig has been announced by GLB. Their radio also operates on 220 mHz, but can go 19,200 bps, or twice the AEA speed. Initial shipments have been made to several beta test sites so far, when the results of these test sites come in, I would expect these radios to become available.

I should point out that both of the above mentioned radios will be relatively expensive initially. In addition, it should be pointed out that both units operate on 220 mHz. It seems just as the equipment is becoming available, the FCC is considering eliminating enough of the 220 mHz band to render these radios useless. The Amateur packet community should redouble our efforts to persuade the FCC that we NEED these frequencies for digital communications. If we lose a large portion of 220 mHz, I feel our next logical frequency for building a network will realistically be 900 mHz. Most of the population centers of the U.S. already have a large portion of our 420-450 mHz allocation used, with even more pressure being placed on it if some of 220 mHz is lost. The 900 mHz band can be considered microwave, requiring paths to be line-of-sight, making network development even more difficult.

Another effort is being made in Atlanta that is even more ambitious. An amateur of some repute in Atlanta has developed a board-set that (when assembled) will run at 56 kbps. This is designed to create the digitally-modulated RF at a 29 mHz IF, which is then transverted to the band to be used. Some of these boards are being sent out now, with testing to be done in the field. This is a potentially exciting development, providing it does operate cleanly and efficiently. One note on this is that there have been some problem getting a TNC-2 to operate reliably at 56 kbps. If this is true, the shoe is on the other foot, in that the RF hardware signalling speed has surpassed the digital hardware capability!

There is also some movement on HF packet operation. There is a new movement afoot by the ARRL and others to try higher speeds on HF using minimum-shift-keying (MSK) modulation. The ARRL has asked for an STA for testing these newer HF signalling techniques. This will allow higher-speed BBS interlinking over long distances.

It appears that there has been movement over the last year in the RF portion of packet radio. Hopefully this trend will continue, with other manufacturers also jumping into the field. My other hope is that the RF systems that do come into being become compatible with each other. We do not need incompatible RF radios trying to inter-operate.

Packet Digital Hardware

With the proliferation of PC clones in the last year, the those working on developing the Amateur Packet network have been migrating toward the IBM PC and clones. At this point, most of the work being done is on the PC. Two of the boards becoming available to support this work are the PC-100 from Pat-Comm and the Eagle RS-232 board.

PC-100 From Pac-Comm

The PC-100 from Pat-Comm is an 8530 packet board for the PC that I designed over a year ago. It has an 8530 Serial Communications Controller (SCC) on it, which drives either an RS-232 connector, or on-board modems. The board is a half-slot size, and plugs into the PC bus. It can provide two different channels of packet operation, and more than one board can be chained together, allowing a switch (or user for that matter) to have many RF inputs. We in AMRAD have had versions of this board running (sending and receiving frames) for over a year now, so we feel it will contribute a lot to packet development. The PC-100 should be available from Pat-Comm Real Soon Now (RSN).

The latest version of the PC-100 uses one 7910 world-chip modem, and optionally a TCM-3105 CMOS modem. The following is the port map of the standard PC-100:

200-203 hex	Modem Control Latch.
204 hex	8530 SCC Channel B Control Port.
205 hex	8530 SCC Channel B Data Port.
206 hex	8530 SCC Channel A Control Port.
207 hex	8530 SCC Channel A Data Port.
208-20B hex	Interrupt Acknowledge Strobe Port.

Bits 0-3 of the Modem control port set the various operating modes of the 7910 modem chip, while bits 4, 5, and 6 control the TCM-3105 modem chip.

One of the changes Pat-Commhas made is to use an external (to the SCC) oscillator for serial clock generation. This allows them to use the now familiar external divide-by-32 circuit from the still-born AMRAD PAD days, making full-duplex operation much better. Unfortunately, this also makes the PC-100 sometimes inoperable with PC clones in turbo mode.

The problem with turbo-mode clones is that the clock signal driving the clock input to the 8530 is not synchronous with the host CPU, and NOT FAST ENOUGH. The 8530 requires the PCLK (clock) input to be at least 90 percent of the CPU clock speed for proper operation. A PC running at

4.77 mHz does allow the 8530 time to cycle properly, but when running a clone at 7 mHz (for example), the 8530 no longer has enough time to function properly with the CPU. The standard PC-100 uses a 4.9152 mHz oscillator, with faster clocks and higher speed parts optionally available. This optional speed-up throws the problem in the software designers lap (which board, what baud-rate divider numbers to use? etc).

The PC-100 is interrupt-driven only. The original PC-100 used pin IRQ3 standard, with jumpers to optionally move the interrupt to other hardware vectors.

Eagle RS-232 Board

Another board has recently become available to the Amateur packet community. When Eagle computers folded, a company in California apparently bought up a large portion of Eagle's equipment. Eagle had designed a RS-232 serial board for their PC compatible computer that uses the 8530 SCC chip. Someone found out these were available, and the rush was on. This company quickly sold thier stock of functional boards to Amateurs hungry for development tools. Several of us in AMRAD were fortunate enough to obtain one or more of these boards, and as far as I am aware, we were the first to get the board to successfully send and receive packets on the air.

The Eagle board is somewhat similar to an RS-232 version of my 8530 board. It provides two channels of serial data flow, with both being RS-232. The SCC chip can operate in either interrupt mode, or can use DMA with the PC (unlike the PC-100, which is interrupt-driven only). Since there are no modems, one must be provided externally. There are two "gotchas" with the Eagle board. Fist of all, the interrupt channel used is the same one normally used by the hard-disk controller. Secondly, TxD and RxD on the DB-25 for one of the channels are backwards from "normal" (if there is a normal in RS-232). Aside from these two glitches the Eagle board does indeed operate on packet with an IBM PC. There has been enough interest in these Eagle boards that the company that purchased the rights to them is considering making more of these boards.

The port mapping for an unmodified Eagle RS-232 board is as follows:

300 hex 8530 SCC Channel B Control Port.
301 hex 8530 SCC Channel B Data Port.
302 hex 8530 SCC Channel A Control Port.
303 hex 8530 SCC Channel A Data Port.
304 hex Board Control Latch Port.

The Control Latch port is broken down as follows:

Bit 0 DMA ONLY...SCC *CTL/DATA, 1 = Data.
Bit 1 DMA ONLY...SCC Channel, 1 = Ch. B.
Bit 2 DMA Operation Gate, 0 = Non-DMA.
Bit 3 Interrupt Enable, 1 = Int Eanbled.
Bit 4 INTACK pin Strobe.

The Eagle board also uses an external oscillator for baud-rate generation. However? it does NOT have the divide-by-32 circuit required for proper full-duplex operation. The standard frequency used on the Eagle board is 3.6864 mHz. Since most of the components on the Eagle board are NOT socketed, speeding itupmay be much more difficult, making this board harder to use with a clone in turbo-mode.

It would be nice to use the DMA capability of the Eagle board with the PC, but that probably won't be done much for the same reason I didn't design DMA capability onto the PC-100. The PC bus only allows for a total of four half-duplex DMA channels. Of these, one is tied up doing dynamic RAM refresh, and one each is assigned to the hard and floppy disk controllers. This only leaves a single half-duplex DMA channel for us to use (without time-sharing the others, which could be a software nightmare). Both my board and the Eagle board support two channels of operation, both of which are potentially full-duplex. In order to support these boards properly, four DMA channels per board are required. Since we only have one-fourth of that available, I decided not to use DMA at all. Trying to decide which channel would have the DMA versus the others would be too complicated with barely minimal improvement in operation. In addition, the PC bus has left out at least one crucial signal, which prevents placing DMA controllers anywhere except the motherboard (at least without serious kludging).

At this point, the rest of the hardware being used for switch development is based on the PC clones that have taken over the hobby computer market. We are working with standard PC Clone motherboards, Clone power supplies, clone floppy disk controllers with either 3.5 inch or 5.25 inch drives, and a small sized hard disk. The floppies are planned to be used for loading new code or data, or for local diagnostic and maintenance functions, while the hard-disk will be used for on-line data backup, with the actual data stored in memory. At first the switch software will be left in RAM memory, but plans are that once it stabilizes, the code will be loaded into EPROM, along with a ROM sanity monitor and boot loader. This boot loader will be capable of loading new code into RAM either from the disk drives or through a simple AX.25 LINK LEVEL ONLY connection.

The PC packet hardware being worked with so far has been both the PC-100 and the Eagle board, with both operating using the same basic software. As the switch complexity develops, more attention will have to be paid to the individual packet channels. I am guessing this will lead to a new board that will plug into the PC bus and use a DMA channel to communicate with the PC. On this board will be a complete computer system supporting the packet channels up to the Link Layer, and then passing the packets to the host PC for higher-layer processing as necessary.

Other 16-Bit Digital Hardware

Rumor has it that a group of Amateurs in Southern California have developed a new Network Node Controller (NNC) (read switch) based on sixteen-bit technology. They are using the Intel 80186 CPU with all static RAM. The packet channels will be supported with our good old 8530 SCC, of which there are at least two, and I think there will be up to three. Each 8530 will be using DMA for all four operations (Tx and Rx for both channels), which should allow for higher-speed operation. Needless to say these boards are eagerly awaited. The last I heard is that they are nearing beta-test shipment.

Eight-Bit Networking

The last vear has seen interest in the eight-bit world almost completely dry up. With the cost of PC clones coming down, virtually everyone working on network development has abandoned the eight-bit systems (some out of necessity, you can only fill a five-pound bag so full). I am afraid the Xerox 820 and the TAPR NNC are two casulties of this sudden swing. I don't think TAPR is going to be doing much about the NNC in the future, and few people are planning to work on the 820 any longer, except as individual user stations or packet BBS systems.

The one area where this is not true is surprisingly the TNC-2, and it's various clones and semi-clones. When it comes to these devices, there are two opposite trends happening over the last year. The first is to provide more sophisticated software inside the TNC-2, both for the end-user and potentially as a true packet switch. Witness the use of the TNC-2 with NET-ROM which turns the TNC-2 into a single channel packet switch. In order to operate a dual-frequency packet switch, two TNC-2 s are connected back-to-back. Also, once a TNC-2 is used with NET-ROM, it can no longer be used as a TNC, there is no support for both operations.

In the same vain, the virtual-circuit networking crowd is still planning to provide ALL end-user networking within the TNC-2. This way the Amateur packeteer does not need an IBM PC or clone just to gain access to the network. In addition, the virtual-circuit crowd is still planning to base a simple VC switch on the TNC-2 for regions that do not require the horsepower of a PC clone, similar to what Howie Goldstein, N2WX showed at the Conference last year.

At the opposite extreme we have the TCP/IP crowd. They are taking the TNC-2 and gutting out all of the AX.25 and user interface, leaving behind what they call the "KISS TNC". The reasoning behind this is that their protocol software barely runs on a PC, it does not run on a TNC-2. They are running packets using unconnected Link Level frames (using only UI frames) with the datagram information packed into the Information field of the UI frame. They rely on the Transport Layer (in this case TCP) to handle ALL error recovery. This "Keep-It-Simple-Stupid" operation allows the PC software to handle all aspects of packeting, except actually generating the envelope around the datagram, which is left to the TNC, along with gaining access to the RF channel. The datagrammies feel this provides much more flexibility, since they can now run unconnected datagram service at both the link AND network layers. This author does not necessarily agree with them, and somehow the picture of eight ganged-together digipeaters from Washington DC to Boston (with 5 minutes and 3 disconnects/reconnects just to say hi) starts painfully coming into focus from the recess of my mind, but that's life. A square wheel is better than no wheel, I guess.

Packet Switch Software Development

While others have been forging ahead designing and writing packet switches, AMRAD has been taking a somewhat slower approach to this task. I personally feel **that** for the packet switch to work efficiently many software design issues must be decided BEFORE the first line of code is written. I have been hearing a lot about writing ALL code in C versus Pascal versus Assembler versus whatever languageof-the-month comes up. My opinion is that there is no ONE SINGLE answer to the language issue. There have been two instances where a software designer/programmer has written virtually all software in a higher-level language (including almost all the interrupt processing). When the software didn't run fast enough, this programmer claimed the hardware was not capable of running any faster, and the only answer was DMA operation. In fact, the hardware could run MUCH faster if the interrupt processing had been done in assembler instead. The point of this is that the software should be optimized for the task. Taking 2 years to write a full switch in assembler can be just as bad as the above mentioned example. Software language optimization for each specific task should be done before proceeding with that task. In addition, some attention should be paid on standardizing how tasks are supposed to intercommunicate, and how shared resources are to be requested and allocated.

Last year I wrote a paper on how I felt the various functions of the packet switch should be broken down. Since that paper was published, I have felt that most of the comments made there are still valid, in fact I feel even stonger that the support services should be standardized within the switch. We in AMRAD have been passing around a lot of ideas regarding switch softtware design, and while not a lot of code has been written so far, I feel much better about the direction we are taking than if we had just bolted off into the ether writing madly away.

Some of the modules I have identified over the last year are as follows:

Shared Resource Modules

SOS	Switch Operating System.
BQM	Buffer and Data Queue Manager.
CTM	Clock and Timer Module.
ECR	Error Control and Recovery.
DBI	DOS and BIOS Interface module.

Internal Switch Modules

LCI	Local Console Interface.
SCC	Switch Command and Control.
AUT	Authorization Module.
UDB	User Database Manager.
RDB	Routing Database Manager.
RTB	Round Table Operations Manager.
PAD	Packet Assembler/Disassembler

Protocol Machines

SSM	Switch Session Module (Layer 5).
ETM	End-Point Transport Module(Layer 4)
NET	Switch Network Module (Layer 3).
LINK	Switch Link Module (Layer 2).
LARM	Link Address Resolution Module.
PHS	Physical 'Hardware Support Module.

Each of these modules has their own distinct function, with attendant trade-offs in usage/size/speed/complexity. Therefore, I feel each module should be studied fairly independantly before deciding how it should be written, and in what language.

Switch Operating System (SOS)

Since the Switch Operating System is responsible for almost all inter-task communication, it will be used a lot. Because of this, it should be lean and mean and not cause much additional delay 'between processes. I feel in the long run, the SOS should be written in assembly language for each target processor.

The operating system we have decided to use is a fairly simple one, often called the HUB operating system. Mike O'Dell wrote a paper for last years conference which describes it in more detail. He also wrote a version of the HUB Operating System in C, which we have had running both on Xerox 820's and IBM PC's. Despite what I said above regarding assembly language, since he already has it working in C, I think we will use **his** version, at least until one of **us** has **time** to re-code it in assembler.

Buffer and Queue Module

The Buffer and data Queue Module is responsible for handling both the data buffers and the data buffer queues. Most of the actual switch processing on received/transmitted data will be done inside these data buffers. In order to provide smooth data buffer flow between processes, data queues will be implemented.

Once again, since the Buffer and Queue Manager is controlling a valuable shared resource, this module should be optimized for speed, implying assembly language. Getting memory for a data buffer from a C language heap each time you need a buffer can take up a lot of extra valuable time.

I have implemented the Buffer and Queue Manager for the IBM PC in assembly language, and this code has been checked out and de bugged. Incidently, it has also been checked out and works properly when called from C routines.

Clock and Timer Module

The Clock and Timer Module is responsible both for maintaining the time-of-day functions and the timer requests from other modules, such as the protocol machines. Once again, it is a shared resource, and should probably be written in assember, to reduce routine latency.

We have decided both the time-of-day and the timer routines should keep a linked list of timer requests, with each request going in order of time. This means only the shortest timer actually needs to be counted down, and once it expires, the requesting routine gets notified, and the next shortest timer becomes the active request. This method is much more efficient than trying to update many timers at each "tick" of the system clock. We are just starting work on **this** module as of this writing.

Error Control and Recovery

The Error Control and Recovery Module will be responsible for either trying to "heal" minor errors, or disable the switch if major malfunctions are encountered. It will also be responsible for accumulating totals for "soft errors" that may occur during normal switch operation, for later software tweaking. This module is not really necessary at first, and can be written in a higher level language such as C.

DOS and BIOS interface

This module will be used to gain access to the various hardware available in the PC, primarily the disk drives. It is not crucial for switch operation at the *beginning, and will probably be written in a higher level language, such as C.

Local Console Interface

The Local Console Interface will be responsible for allowing local access to the switch either for "tweaking" or for maintenance. Some form of this module will most likely be required when the switch is first activated, but it will probably be expanded upon as the need arises. Once again, it could be written in a higher level language.

Switch Command and Control

The Switch Command and Control module will be responsible for interpreting and acting upon the commands given to it either by the Local Console Interface, or by a packet connection to the switch via the Authentication Module. Obviously, some form of this module must be operational to alter operation of the switch. It has not been written, and will most likely be written in a higher level language.

Authentication Module

As described by Hal Feinstein at last year's conference, some sort of message authentication must be provided if the switch internals are to be accessed via packet. Hal has been working on this aspect of the switch over the last year, with most of his effort being written in C.

User Database Module

The User Database will become increasingly important as more users show up on the network. This module has received almost no attention at this point. It is not needed for basic switch operation, but will be added at a future date. It could be written in a higher level language.

Routing Database

The Routing Database is used by the Transport and Network protocol machines to get the proper information on network connection routing at call setup time. This database is verp important for automated network routing. envision the actual database being held in RAM memory, with periodically disk compares/saves for backup purposes. This software has not been written yet, and depends on routing and naming conventions not yet decided upon. It could be written in a higher level language, since connection-oriented networks only need to access it at call setup time, not on every packet like most datagram networks.

Roundtable Module

One of the neat functions Howie N2WX provided in his second network level software was something that has been sorely lacking in Amateur Packet Radio, the roundtable. It allowed for multiple stations to connect to the switch, and all of them talk to each other, ala a voice roundtable. I have felt for a long time this was needed and was glad Howie showed it could be done. This module has not been worked on at this time, but will probably be written in a higher level language.

PacketAssembler/Disassembler

One of our firm committments is to make our switch downward compatible. In order to do this, we must provide some method for older, non-network capable TNC's to gain access to network functions. Once again, Howie showed how this could be done in his first release of network code for the TNC-2, as described in a paper presented at last year's conference. This is another task easily written in a higher level language.

Switch Session Module

The Switch Session Module will be responsible for handling the various internally terminated network connections. These include connections to the Switch Command and Control Module, the Roundtable Module, and the PAD module. It could be written in a higher level language.

End-Point Transport Module

The End-Point Transport Module will be responsible for assuring the user data made it through the packet network without errors or problems (if that was requested). It is based on X.224 Version TP-1, and has been described in other papers at the last two conferences. I feel it could be written in a higher level language.

Network Protocol Machine

The Network protocol machine will use the X.25/X.75 Network Layer protocols. Some background work has been done on writing this module, but no code has acutally been written. I feel the Network layer could be written in either assembler or a higher level language such as C.

Link Layer Protocol Machine

The Link Layer Protocol machine has been started. It is based on the AX.25 Level 2 protocol, and is being written in assembler. I feel the Link Layer protocol machine has enough time-crucial elements that to implement it in a higher level language would add too much delay, especially since we are starting to see higher speed packet operation.

Link Address Resolution Module

I have wedged a module in between the hardware support and the Link layer protocol machine, which I call the Link Address Resolution Module. It will take our Amateur callsign address kludge out of the received packets before passing them to the link module, and add the kludge to all transmitted packets from the link module. In addition, it will handle the digipeating function instead of the Link module. This separation will allow the Link Layer protocol machine to be more of a generic X.25 machine, reducing it's complexity.

This module is being written in assembly language, and is almost finished. This will allow testing of the switch shared resources modules as a simple digipeater until the Link, Network, etc modules are finished.

Physical Hardware Support Module(s)

The Physical hardware used to send and receive packets must have some software to support it. This is the job of the Physical Hardware Support Module(s). As implied, there can be more than one of these modules. In fact, there will be one of these modules for each hardware board in the switch. So far, both of the boards being used in our switch development use similar hardware (the 8530 SCC), so the PHS modules for these boards are also very similar. As stated earlier in this paper, both the boards have been on the air sending and receiving packets. All that is necessary is to modify the software drivers slightly to interface properly with the shared resources modules. The PHS modules are written in assembly language for the PC, using interrupts.

As can be seen from both the above description, and my paper from last year, our effort at developing a packet switch is probably more detailed and studied than most other implementations. I feel this will aid both ourselves in the development process, and others who follow after us in understanding what it takes to implement a complex function such as a packet switch.

Off On a Tangent

1 thought I might bring up something that has nothing to do with any of the above, but I feel should be mentioned somewhere (since I don't have

time to write another paper on it, here we go). I have noticed that a large portion of the packet activity revolves around packet bulletin boards. I generally agree with a good part of this, but I would like to point out something that might save quite a bit of channel activity in the long run. If more Amateurs were to put up their own bulletin board systems rather than using a central, community BBS, then other Amateurs could send mail directly to them, rather than a BBS they happen to frequent. This will lead to a reduction of overall packet activity, since the message can be automatically forwarded directly to the destination Amateur, reducing the chances of being read multiple times, and showing up on every message summary request. Something to think about for now.

Conclusion

There has been some movement over the past year in packet hardware, both digital and RF. Admittely, after the initial surge last year by the Virtual-Circuit crowd, the Datagrammies have caught up. I hope to have more time over the next year to finish our connection-oriented packet switch project, and start placing them into service. I still hope for an experiment where both approaches are placed side-by-side for a period of time and compared. I still feel that the Virtual-Circuit approach will be the long-term winner by a wide margin.

References

Fox, T., "RF, RF, Where is My High Speed RF?", 5thARRL Computer Networking Conference, ARRL, 1986

Fox, T., "Packet Switch Software Design Considerations", 5th ARRL Computer Networking Conference, ARRL, 1986

Fox, T., "User and! Switch Packet Digital Hardware", 5th ARRL Computer Networking Conference, ARRL, 1986

O'Dell, M., "An Introduction to the HUB Operating System", 5th ARRL Computer Networking Conference, ARRL, 1986

Feinstein, H., "Authentication of the Packet Radio Switch Control Link", 5th ARRL Computer Networking Conference, ARRL, 1986

Goldstein, H., "A Packet Assembler/Disassembler for Amateur Packet Radio Networking", 5th ARRL Computer Networking Conference, ARRL, 1986

Beattie, G., "Proposal: Recommendation AX.224, Class 1 Transport Protocol Specification for the Amateur Radio Network"., 5th ARRL, Computer Networking Conference„ ARRL, 1986

Fox, T., "CCITT X.224 Transport Layer Protocol basic Description", 4th ARRI, Computer Networking Conference„ ARRL, 1985

Rinaldo, P., "Introducing the Packet Adaptive Modem (PAM)", 2nd ARRL Computer Networking Conference, ARRL, 1984

Fox, T., "A New Packet-Radio-Controller board", 2nd ARRL Computer Networking Conference, ARRL, 1984

Thoughts on the Issues of Address Resolution and Routing in Amateur Packet Radio TCP/IP Networks

Bdale Garbee, N3EUA

ABSTRACT

The current KA9Q software includes a technique for automatic address resolution, but does not include automatic routing features. The difference between routing and address resolution is explained, and several concerns for on-air automatic routing algorithm implementations are mentioned.

Background:

One of the major goals of those involved in the TCP/IP project begun by Phil Karn KA9Q has been to develop a network implementation that automatically handles routing. By this, we mean that it should not be necessary for an end user to have any knowledge of the topology of the network he is using in order to perform normal "day-to-day" network activities. The user's involvement should extend only to requests of the form "get file from host < n >" or "send mail to user <n > at host <m >".

Several groups are currently working on software that implements one or another form of automatic routing. The NET/ROM product requires that the user only know the nearest NET/ROM site to both the originating and destination stations in order to establish a circuit between the two. The TEXNet system reportedly handles traffic between its endnodes in a similarly transparent fashion. But in both of these cases, the user must still be aware of some aspects of the network topology above and beyond knowing the hostname of the destination site.

What we have right now isn't *good enough!*

ARP, the Address Resolution Protocol:

In his TCP/IP package, Phil Karn has included an implementation of the ARP Address Resolution Protocol, which was originally created to allow for automatic mapping of 3%bit IP addresses into 48-bit Ethernet addresses on a local wired subnet. ARP works by having the originating station issue an "ARP Request" whenever it wishes to converse with a host whose IP address is known, but whose "physical address" is unknown. An important thing to remember about ARP is that it was designed for subnets where all hosts can hear each other directly.

In the Amateur Packet world, the physical address is equivalent to the destination station's callsign and optional sub-station identifier, or SSID. If and when the destination station hears the broadcasted request, it knows the IP address and physical address of the sending station by examining the contents of the request packet, and it responds by sending an "ARP Reply" packet directly to the originating station.

The value of ARP is that the only piece of information about a station that you need to know ahead of time in order to make contact with them is their IP address, as long as they are within RF range. ARP performs very well at the task it was designed for, which is simple mapping of logical address into physical address.

The State of Routing:

There are two simple situations that demonstrate the limitations of ARP, and will serve to distinguish address resolution from routing. Let us consider the IP Switch, and the Digipeater.

A digipeater is a device that receives a packet that has been source routed through it, and echoes that same packet on the same or a different channel, changing the packet only by marking it as "having been

digipeated by **this site".** Our interest in digipeaters stems only from their current popularity, and resulting availability. In order to make use of a digipeater with the KA9Q Internet software, both the source and destination stations need to explicitly provide the physical "digipeater string" information to the NET.EXE package, as ARP is totally unaware of the existence of digipeaters. This is not good. Several extensions to ARP **have** been proposed, but **a** better solution than extending ARP may be to manual routing of digipeaters for IP, and rely on IP Switches and/or other "level three" network. implementations to make the subnet appear to be contiguous.

An IP Switch in a certain sense performs the same function as a digipeater from the end-user viewpoint. The Switch receives packets that are destined for other hosts, and forwards them via one of perhaps several channels. For those familiar with Ethernet or other wired networks) running TCP/IP, an IP Switch in the Packet Radio environment differs from an IP Gateway primarily in that a Switch **may** want to retransmit a packet on the same channel it heard it on, while a Gateway would never want to do that. The fundamental difference between a digipeater and a Switch is that a Switch makes routing decisions, while a digipeater depends on the end user to source-route **a** connection through it.

Address resolution therefore is the mapping of a "network address", such as an IP address in the TCP/IP environment, into an actual physical station identifier, such as a TNC callsign. Routing is the set of issues and algorithms that surround passing packets through intermediate stations,,

Thoughts About What We Need:

Routing individual datagrams is currently handled by manually inserting entries into a station's "route table" that tell the software to route all packet for a given subrange of IP addresses via a specified gateway, or switch. This is already a very powerful mechanism, in that a user can specify his default routing to be via his local switch, and then add special cases to "un-default" local stations he can hit directly, and the end user problem is therefore greatly reduced.

But the IP Switches currently run the same software. The only way to build a network is to manually create routing entries at each switch. We need to develop an algorithm for automatically determining reasonable paths, that can automatically update the routing tables at each Switch site. There are two different philosophies about how to do this. One is based on the idea that the network should "already know" what to do with each packet when it arrives at **a** given switch location, the other is based on the idea that someone requesting a connection to a site that is not known locally generates by their request the initiative for the network to go figure out how to get there.

The tradeoff at the highest level is simple. The first mechanism forces the network to continuously be passing information about routes between switches, all of which is network overhead. The second mechanism generates a great deal of traffic on the first request for **a** path segment that does not currently exist in **the** routing tables, but probably **has a** lower overhead overall. It does, however, cause a much greater "turnaround time" delay for the user requesting an unusual route. What the best solution to this tradeoff between overhead and individual user performance is has not in my opinion been sufficiently considered.

A final thought for this paper is that we need to be able to handle mobile stations, not just stations that are away from their home switches, but stations that are actually in motion. This probably cannot be done by schemes that dynamically reassign IP addresses for the simple reason that a station might have a connection in progress when it crosses a switch boundary. Food for thought.

Related Issues:

There are two other issues that need to be considered. One is the mapping of mnemonic hostnames into IP addresses, both for clients like FTP and Telnet, and for Mail transfer. Secondly, there is **a** separate but similar routing problem that exists in the mail handling portion of the network software, in that a given node may have more than one mail transport system available to it (including perhaps things like uucp and Fidomail), and must make a "routing decision" as to which delivery agent should be called to process **a given message.**

The current means of handling the mapping of mnemonic hostnames into IP addresses is to maintain **a** file listing the possible translations that are known to the system, and requiring the user to specify the IP address directly when a host entry cannot be matched. A much better system would be to designate a host-name server in each local subnet, and have the client software negotiate with the name server to map

a **hostname** into an address. This reduces by far the number of locations that must maintain a full **host**-name database. **Cacheing** of some number of most recently or frequently used addresses on the local machine should minimize the impact of name-serving on the network's overall performance.

Mail routing is a fascinating topic, but one which deserves much more room than that which is available here. Suffice to say that the problem of mapping a given mail message into an appropriate delivery agent on a given mail-handling host can be as simple or as complicated as routing packets at a packet switch.

Wrong Things We Could Do:

With amazing regularity, well-meaning hams have suggested a variety of schemes for "routing" packets based on everything from grid squares to zip codes, to telephone exchange prefixes. The fundamental problem with this is that there is no relation between ***a host's name or address and what is the best path to take*** to ***reach that*** host. A classic example is the satellite "wormhole" between CA and MD, which often means that to get from the **midwest** to the east coast, the best path would be to go west, not ***east,*** to the **wormhole** end, and tunnel across country.

We need to design automatic routing mechanisms that do not rely on host names or network addresses. We can use names and addresses as hints, but cannot expect them to always work, particularly given the mobile nature of the modern ham! Routing depends on a network's topology, not the geography or political boundaries that it crosses. Any routing mechanism that we adopt must handle mobile stations as well, which is virtually impossible when the routing mechanism is based on some fixed geographical reference system.

Another interesting proposal has been to not assign fixed IP addresses to individual stations. There are some schemes for dynamically assigning addresses to small hosts in use in major universities, but I am not yet convinced that this is a good thing to do on packet, primarily because our environment poses some additional complications over those present in the wired university environment. The most obvious of which is the mobile station, which might have a connection in progress as it crosses an address allocation zone. None of the existing schemes that I am aware of allow for mobile stations. We should, however, keep an eye on progress in this direction outside of amateur radio, in case someone makes it work.

Conclusion:

The purpose of this paper has been to document some goals for those implementing routing mechanisms for packet radio, and to raise some specific concerns that should be understood and considered. Much work remains to be done. Many of the ideas discussed here are under consideration by the group of people working with the **KA9Q** Internet software, and undoubtedly we will eventually see an automatic routing implementation either as part of the package, or adopted from a third-party source's level three implementation.

References:

1. Tanenbaum, A.S., "Computer Networks", Prentice-Hall, 1981.
2. RFC826, Address Resolution Protocol.
3. **Beattie, J.G.,** "Proposal: Recommendation **AX.121NA** Numbering Plan For The Amateur Radio Network in North America", ***Fourth ARRL Amateur Radio Computer*** **Networking Conference,** 1984.
4. **Beattie,** J.G., "Proposal: Recommendation AX.121 International Numbering Plan For The Amateur Radio Network", ***Fifth ARRL Amateur Radio Computer Networking Conference,*** 1986.
5. Fox, Terry, "Amateur Network Addressing and Routing", ***Fifth ARRL Amateur Radio Computer Networking Conference,*** 1986.
6. NET/ROM Beta-Test Preliminary Documentation, Software 2000 Inc.
7. **TEXNet** Presentation at the 1987 TAPR Annual Meeting.

The Design of a Mail System for the KA9Q Internet Protocol Package

Bdale Garbee, N3EUA
Gerard van der Grinten, PAØGRI

ABSTRACT

The current implementation of the mail manipulation system that has been built by N3EUA and PAØGRI for the KA9Q Internet Protocol Package is described briefly. A proposal for the next generation of network mail handling for the KA9Q Internet environment is described. An important change in the way we can and should think about electronic mail in the amateur packet radio world is discussed.

Background:

Since the conception of "Amateur Packet Radio", quite a bit of evolution has occurred. Many lessons have been learned the hard way, and many are yet to be learned. However, one point which has become increasingly obvious to those of us involved in the development of packet radio software, is that a much more consistent and powerful way of dealing with electronic mail messages is

In the beginning, there was raw data transfer from point A to point B without any protocol, and it was good. Then came a protocol called AX.25, which allowed for error-free point-to-point virtual circuit links, and it was better. Most recently has come an implementation of the DARBA TCP/IP protocol family, and it is the best so far. The fact that this protocol family includes a defacto real-world mail-handling protocol, SMTP, has caused some interesting changes in the way we think about mail in packet radio.

The current state-of-the-practice in amateur digital mail handling is for one user, or ham, to "log into" a local PBBS system, by establishing a virtual circuit to the PBBS's user interface. He can then enter, read, or delete a given message, and then "log off". Whenever users are not connected to the PBBS, it is available to attempt "forwarding" of the mail, a process in which "packets" consisting of entire messages are routed from one PBBS to another. Once the message has arrived at the destination system, the end user or ham can read the message using the same "connect to the PBBS" sequence that the sender used.

There are some fundamental design problems with this process. It works, and has solved. the first part our mail-handling needs. But it requires that both the sending and receiving participants spend a non-trivial amount of their time interacting with a remote user-interface, which implicitly includes waiting for channel throughput on the AX.25 circuit. In addition, there are very few services available to the user, such as message archiving or printing, message forwarding, editing of messages, etc. It is also the case that some mechanism (usually beacons) must be provided to allow the receiving mail user to know that there is a message waiting for him on his local PBBS.

It is interesting to note that these limitations are well-known and understood in the packet community. An interesting trend has been the creation of programs to allow machines such as the Tandy 100 and C64 to serve as "mini-PBBS" systems, or "personal mailboxes". Moving as much of the mail user interface from a remote system to the user's local computer as possible provides several benefits. The user need no longer suffer the delays associated with packet flow on the channel while interacting with the mail system, and it becomes possible to provide features such as message filing, printing, and more extensive reply and forward capabilities. While this may not have been possible in the early days of packet radio due to the use of dumb terminals as packet stations, almost all packet radio stations today include a computer as the end terminal.

While a transfer of the mail user interface from a PBBS to the user's local machine is in theory possible with any underlying network protocol family, the recent development by **KA9Q** and others of a complete implementation of the DARPA TCP/IP protocol family, including the defacto-standard mail transfer protocol SMTP, has hastened the need for such a transfer. Implementing a rational mail user interface and underlying servers for the **KA9Q** Internet Package has been a most challenging task, with the resulting system's design principles and implementation tradeoffs occupying the remainder of this paper.

The Current Implementation:

The current mail system in the **KA9Q** Internet package consists of an SMTP server and SMTP client that are integrated into the NET.EXE package, and a separate mail user-interface. The mail user interface, while currently sporting a very terse user interface, provides facilities for creation and reading of messages, and some crude message filing capabilities.

When a user wishes to send a mail message, he runs the mail user interface program ***BM,*** and enters the destination address, subject, and text of his message. The message is queued in a directory on his machine for later transmission. At regular intervals, the SMTP client in NET.EXE scans the queue directory and attempts to process waiting outbound mail messages. Each message is delivered by the client directly to the server at the destination station.

When an incoming message arrives at the local SMTP server, it is appended to a mailbox file, the name of which is based on the **username** given in the destination address. Multiple real or pseudo-users (for mailing lists) are therefore supported on a given machine. At his leisure, the user can run the ***BM*** user interface to read and manipulate the messages in his mailbox file(s), and/or can manipulate them directly as disk text files.

The biggest change from existing PBBS mail handling is that the user never has to wait for a packet to be transferred on the channel. All on-air activity is handled automatically "in the background".

Environment:

As may be obvious, the design concepts and implementation ideas presented here are applicable regardless of the particular computer facilities in use, and/or the specific low-level transport protocol and mechanism available. However, for the sake of comparison and reference, we will describe briefly the hardware and software environment under which our mailer implementation has been developed.

The **KA9Q** Internet Package has been developed on the IBM PC, and close compatibles. It uses the Intel Small Memory Model, which allows 64k of code space and 64k of data space on the 8088 processor. In an effort to avoid the necessity of going to a large memory model with the attendant segment maintenance overhead on the 8088, and as an aid to generality and portability, functions such as the mail user interface are being developed as separate applications, designed to run concurrently with the **KA9Q** Internet Package using a PC multitasking package such as Doubledos. In the future, the ideas and algorithms presented here will most likely be implemented under Minix/Unix as well.

Hardware requirements are essentially the same as for the **KA9Q** package itself, except that mail handling can frequently consume considerably more disk space that is required for the protocol package to run, meaning that a large forwarding site will need a hard disk. A simple single or dual user PC should be able to get by on one 360k floppy drive.

All of the mailer software, as with the rest of the **KA9Q** package, has been written in C using the Aztec MS-DOS compiler. It is therefore extremely portable, with the exception of a small module of PC-specific file-system oriented functions, which should be easily translatable to other operating environments.

Top-Level Design - Our Goals:

The next step in the development of our mail system is partitioning of the mail handling tasks. We choose to separate our operations into four distinct categories: user interface, routing, reception, and delivery.

The user interface level is fairly obvious. This is the hunk of code that the user actually sees, and which allows him to enter, read, print, file, delete, forward, etc., all of the messages he must deal with. There can and should be an ability to support multiple user interfaces, to deal with varied and changing user needs, hardware availability, etc.

In the middle, a "routing agent" receives requests from user interfaces, and determines the appropriate target address and delivery agent for each message. It is the only place in which knowledge about the mapping between mail addresses and mail delivery techniques should exist.

At the other end of the scale are the (potentially multiple) mail delivery agents. These are the functional blocks designed to take a message in a reasonable format (for them), and actually do the work of deliver-, ing it. In the SMTP world, this might entail opening a TCP connection to the destination host, and engaging in the SMTP protocol to deliver the message. In the UUCP world, this might include looking up and dialing an appropriate telephone number, performing the login dialog, and then engaging in the required uucp protocols (similar to the way uucico works under Unix). In the Fido world, this might simply be a matter of reformating the message file into a Fido- compatible message, and moving it to an appropriate directory on a PC's hard disk.

The final class of entity is the mail reception agent. Each server receives inbound messages from one of potentially many mail services and puts them in the routing agent's queue, in the same manner that a user interface queues a new message for processing.

The flow of information therefore includes an influx of messages to be handled, both locally created by the user and his interface, and remotely created and received by a server. All pending messages pass into a queue and are processed by the router. The router places each message, potentially with some additional delivery information, into the queue associated with a particular mail delivery agent. Each delivery agent watches its queue for messages to process, and attempts to deliver them.

The Core of the Implementation:

Some simple observation of the mail management system that we have thus far described should reveal the fact that a considerable amount of work will be done enqueueing and dequeueing messages in a set of queues, one for the routing agent and one for each delivery agent. To allow for a consistent means of handling queue'ed messages, a combination of file system usage and custom databasing routines has been laid out.

In the most general case, a queue entry will consist of two files, one for the text of the message (most likely in RFC822 format), and one for the "work" information required by a particular delivery agent, or the router. A very reasonable idea is to force each of these files to have a completely unique name. Thus, the primary task of the database is to manage file naming, and connection of a given text file with it's associated work files. Exact details of the database are the topic of current design effort as of this writing.

The routing agent is an extremely important part of this design. It must "read" each message and make some determination about which delivery agent should be used, and what address the delivery agent should try to deliver to. In some situations, such as forwarding uucp mail, which is frequently source-routed, there is information external to the message body that will allow the router to '"cheat". In the majority of cases, however, the router will have to deduce the appropriate agent and address by reading the To: address field in the RFC822-format message body, and comparing it to patterns present in a table of heuristics.

Conclusion:

The intent of this paper is to document the current status and future plans of those working on the mail user interface associated with the KA9Q Internet package. Because a great deal of the detail about how the system is and will be put together is still in a state of flux, it would seem inappropriate to discuss that detail here. Those interested in following the progress of, and/or assisting in the work remaining in this project, are directed to contact one or both of the authors.

References:

1. RFC821, Simple Mail Transfer Protocol.
2. RFC822, Standard for the Format of Internet Text Messages.
3. Allman, Eric, "SENDMAIL - An Internetwork Mail Router", supplied with the 4bsd Unix manual set.

A BIT ERROR RATE TESTER FOR TESTING DIGITAL LINKS

STEVE GOODE, K9NG
140 W. WOOD APT. 314
PALATINE, ILL. 60067

Introduction

With the availability of several link-level protocols such as TEXNET and NETROM, many packet groups are installing inter-city digital links. As AEA and GLB begin shipping their digital radios and the 56 kbps radio demonstrated at the Dayton Hamfest becomes available, more Amateurs will require a method of testing the performance of these digital radios. This paper presents a test circuit which can be used during initial installation of a digital link to enable the installers to fine tune the radio on site. This circuit temporarily replaces the TNC or control computer and allows the installers to listen to the data link to determine how well it is performing. It can also be used for bench testing digital radios and modems.

Bit Error Rate Testers

The performance of any digital radio receiver is tested by measuring its bit error rate (BER). The BER of a data system is the probability of not receiving the transmitted bit correctly. This is normally expressed in percent or decimal form. For example, a, system with a BER of 1×10^{-3} or 0.1% has the probability of receiving the transmitted bit incorrectly once in every 1000 bits. Bit error rate is measured by comparing the transmitted data bits with the received data bits and counting the number of bit errors.

A block diagram of a straight-forward bit error rate tester (BERT) is shown in figure 1. A pseudo-noise (P-N) generator is used to generate a random data stream which is then sent to the modulator and over the channel to the demodulator. The P-N data is also sent to a delay box. The delay box is used to line up the data received from the demodulator with the transmitted data. Every modulator, channel and demodulator will have some delay due to the filters used in each. The transmitted and received data streams are then compared and the errors counted in a counter. Common implementations of BERTs use an XOR gate as the comparison elements which gives a high output when an error occurs. In order to count two errors in a row, the XOR output is usually ANDed with the data clock to give a transition to the counter for every error made. The delay element is usually made up of D flip-flops which are clocked at some multiple of the data clock. The higher clock allows for incremental adjustment of the transmit data stream since the delay of the data system may not be in whole bit increments. The P-N generator is probably the most unfamiliar part of a BERT to most Amateurs. This circuit is similar to an oscillator in a radio. As in an oscillator, the P-N generator has a feedback element which has delay and produces the P-N data stream. In digital circuits D flip-flops are used as delay elements.

A- block diagram of a P-N generator is shown in figure 2. The delayed data is combined in an XOR gate and the resulting bit is sent back to the input of the delay stages forming the feedback path. The data rate of the P-N generator is determined by the clock which clocks the delay stages. A P-N generator is called a pseudo-noise generator because it does not exactly produce a true noise output since once a P-N generator is started it is possible to determine the bit pattern it will generate. BY selecting a P-N generator with a long bit pattern, a sufficiently noise like output will be generated. The length of the P-N pattern is determined by the length of the delay element in the P-N generator. The length of the generated bit pattern will be (2 to the N) -1 where N is the number of delay elements (D flip-flops). For example, a P-N generator with 17 stages will produce a bit pattern 131071 bits long. At 9600 bps it will take over 13 seconds to repeat this pattern which is a long and suitably random pattern for any BER tests we may wish to do.

Although the BERT shown in the block diagram of figure 1 is excellent for bench tests, it has one major flaw which excludes it from being used to tune up links. That is it requires the transmit data stream to compare to the received data stream in order to count errors. What is needed is a way of transmitting random data over the link but still have a method of counting the BER at the receiver without the knowledge of all the individual transmitted bits.

An Alternative BERT

BY adding an XOR gate to the P-N generator as shown in figure 3, it can be seen that the bit pattern generated by this circuit is the same as that generated by the original P-N generator since a low on one input of an XOR gate will give the same data out as put in. If the low is now removed and that XOR input is connected to another data stream, the resulting circuit is a randomizer which is used in many modems to scramble-up the data in order to reduce the low frequency content of the data being sent. In other words, a P-N generator can also be looked at as a randomizer with just a one or a zero for the data input. This means we can send a P-N data stream over the test link and at the receiver build an un-randomizing circuit to get the original one (or zero) back again. We can then count the amount of zeros (or ones) we receive at the output of the un-randomizer and have the BER of the test link.

A block diagram of the un-randomizer is shown in figure 4. The un-randomizer contains a delay register the same length as the P-N generator and also contains an XOR gate, but the circuit is connected in an open loop configuration instead of a feedback loop. To show that the un-randomizer does indeed give a one output when a randomized one is being sent over the test link, we can go through the equations of the randomizer and un-randomizer. If we say the output of the Ath delay element of the randomizer is R(A) and the output of the Bth delay element is R(B). Then the output of the randomizer is:

$$R(A) \oplus R(B) \oplus 1 = R$$

If we say that the channel data bits are denoted by C, the Ath delay channel bit at the un-randomizer is C(A), and the Bth delay channel bit is C(B), then the output of the un-randomizer is:

$$C(A) \oplus C(B) \oplus C = D$$

We can see that C=R since the randomizer delay elements are just delaying the channel data bits. Therefore, at the output of the un-randomizer:

$$R(A) \oplus R(B) \oplus R = D$$

If we replace R with the output of the randomizer we get:

$$R(A) \oplus R(B) \oplus R(A) \oplus R(B) \oplus 1 = D$$

Since any data XORed with itself is zero, we see that D = 1.

BY similar arguments, we can prove that if a zero is randomized a zero will be output from the un-randomizer. We can also show that if the data in the channel is inverted, the resulting un-randomizer output will be inverted from the original data sent.

A schematic of the resulting BERT is shown in figure 5. U1 is a bit rate generator which contains an oscillator and dividers to obtain several data rates. If you are only going to operate at one data rate, you may want to replace this circuit with a fixed oscillator and dividers. A 16 times clock is required for the internal clock recovery circuit. U2 and u3 form the internal clock recovery circuit. In transmit this circuit gives a data clock to the P-N generator. In receive a clock is derived from the receive data to operate the un-randomizer. The receive clock is brought out to a test pin on the pannel to allow the connection of an oscilliscope to observe the receive eye pattern in the modem. An external clock input is proved for in receive for the case where the modem has a receive data clock output. u5 is the delay register for the P-N generator and un-randomizer. A 1.7 stage P-N generator was selected (1). U6 switches the circuit from a P-N generator in transmit to an un-randomizer in receive. U4 provides the XOR gates for the P-N/un-randomizer and also allows the data and external clock to be inverted. U7 provides retiming of the receive data and insures there are no timing problems in the P-N generator design. U8 re-times the un-randomized output. U9 ANDs the re-timed randomizer output with the receive clock to provide an error count output for the counter.

U10 is an audio amplifier. The re-timed output of the un-randomizer is sent to the audio amplifier to allow the data channel to be listened too. When noise is being received at the receiver, noise will be output at the un-randomizer output and can be heard in the speaker. When an error free signal is received, a zero will be put out of the re-timed un-randomizer and quiet will be heard in the speaker. For any error received a pop will be heard in the speaker. Therefore, to tune up the digital link, the installer would tune the radio for best noise quieting just as is done on a FM radio receiving a carrier.

Initial Tests

The BERT can be built in about a night using wire-wrap techniques. Once built, connect a counter to the counter output and set the data rate to 9600 bps. Put the Tx/Rx switch in the Tx position. Noise should be heard coming from the speaker. This is the P-N data being generated. If no noise is heard, put the DATA/m switch in the DATA position and

switch the Tx/Rx switch to Rx and then back to Tx. Noise should now be coming from the speaker. The P-N generator has a problem in that it can lock up in an all ones state. Switching momentarily to Rx puts a zero in the delay register and forces it to start generating a pattern. Switch the Tx/Rx switch to Rx. Put the counter in a one second gate mode and it should read 9600. This is due to the un-randomizer filling up with all zeros and putting out a zero all the time. In one second you will get 9600 bits and in this case they are all wrong since the BERT is expecting a one to be sent. Put the DATA/$\overline{\text{DATA}}$ switch to $\overline{\text{DATA}}$ and the counter should read zero. In this case the un-randomizer has filled with all ones and is putting out a one so no errors are counted. Put the Tx/Rx switch into Tx and make sure noise is heard from the speaker. If no noise is heard restart the P-N source by again switching the DATA/$\overline{\text{DATA}}$ switch to DATA and switching the Tx/Rx switch to Rx and back to Tx. The counter should now read about 4800 since the error output is now looking at the random data that would be sent over the channel. You will not get 4800 exactly every time but if you would average a lot of readings the average should come close to 4800.

A Note on Freauencv Counters

I have used many commercial frequency counters, such as the Fluke 1953A and HP 5335A with this BERT without any problems. Unfortunately, I have also had to modify some counters to get them to work properly with this BERT. As an example, I modified a Ramsey CT-90 by running the error count output of the BERT directly to pin 28 of the Intersil 7216D used in the CT-90.

Using the BERT for Link Tests

For actual link tests you will need two BERTs. One at the transmitter site and one at the receiver site. The receive site should connect to the receive modem and verify that noise is heard with no signal. The transmitter should then go on the air and verify that the P-N generator is operating by listening for noise in the BERT speaker. The receiving BERT should quiet when the transmitter comes on the air. Any pops heard at that point are now errors. Since you obtain instant feedback as to when an error occurs, you can now do tests to see where the errors are coming from or tune the radio for least errors.

Actual BER tests of the link can be done by connecting a counter and counting the number of errors received. The bit error rate can then be computed. The BER is:

$$BER = \frac{\text{Number of Errors Counted}}{\text{Bit Rate X Counter Gate Time X 3}}$$

The factor of three in the denominator of the BER equation is due to the fact that every channel error that comes in is used three times in BERT. It is used once when it first comes in and is then used again when it gets to R(A) and then again when it gets to R(B). This three times error multiplication must be divided out to get the actual channel error rate. The three times error correction is valid for channel error rates of 1% or less.

As an example of calculating the channel bit error rate, if the link is operating at 9600 bps and the counter gate is ten seconds and 288 errors were counted, the BER is 288/(9600 X 10 X 3) or 0.1%.

By listening to the speaker output for different BER, you will eventually be able to have an idea what the BER is just from the speaker output. Using these techniques you can measure the margin of your digital links.

9.6 and 56 kbps modems

Those groups that have the TAPR 9600 bps modem (2) or use the 56 kbps modem shown at Dayton will not have to build all the circuitry shown in figure 5. These modems already contain the 17 stage randomizer and un-randomizer so all that is needed at the transmit side is a 16 times clock and a high input to the transmit data input. At the receive side, all that is needed to listen to the link is an audio amplifier. I have used Radio Shack catalog number 277-10088 amplifier with a 47 k resistor between the receive data out and the audio amplifier input. To actually measure the BER you would need the circuitry shown to the right of the dotted line on figure 5. In calculating the BER for these modems, the three times error multiplication should not be divided out, since this error multiplication is in the modem.

Additional Uses for the BERT

Two BERTs can be used to test the turn around time of digital radios. By observing the data out of the un-randomizer on the receiving BERT on one channel of an oscilliscope and triggering the oscilliscope on the transmit line of the transmitter, the time from scope trigger to the point when all ones are coming out of the un-randomizer is the turn around time of the transmitter.

The receiver recovery time can be measured by transmitting continuously a P-N sequence and observing the output of the receive un-randomizer on an oscilliscope triggered on the release of the transmit line. The time from scope trigger to the point when all ones are coming out of the un-randomizer is the turn around time of the receiver.

The P-N generator can be used as a noise source for doing audio tests as can be heard in the speaker of the BERT. This noise source can be used to test HF modems. Since our present HF modems are audio modems transmitted using linear (SSB) techniques, an audio test using three BERTs could be performed (4).

Conclusions

A bit error rate tester has been described which allows installers of digital links to listen to the quality of the link and measure bit error rate of the link. This BERT can also be used for bench testing future data modems or for more accurate testing of existing modems. Quantitative tests comparing various modems can be done using this BERT.

An attempt was also made to explain the operation of a P-N generator.

Acknowledgements

I would like to thank Bruce Eastmond, K9LAY for thinking of listening to the error output of BERT. Thanks also go once again to Paul Newland, AD71 for providing the state machine code used in the clock recovery circuit.

References

1) S. Golomb, "Shift Register Sequences", Holden Press, 1967.

2) S. Goode, K9NG, "The Bit Error Rate Performance of the TAPR TNC Modem", QEX #18, August 1983.

3) s. Goode, K9NG, "Modifying the Hamtronics FM-5 for 9600 BPS Packet Operation", Fourth ARRL Amateur Radio Computer Networking Conference, San Francisco, Ca., March 30, 1985.

4) G. Kaatz, W9TD, "Modem Modification Tests", Packet Status Register #11, June 1984.

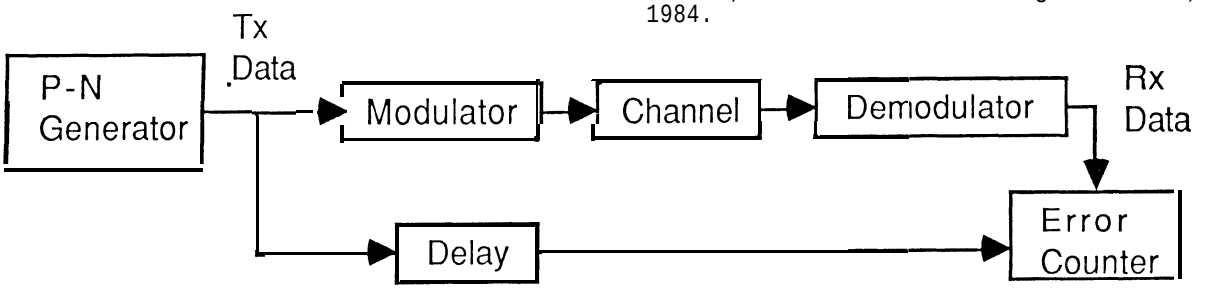


Figure 1

Bit Error Rate Tester

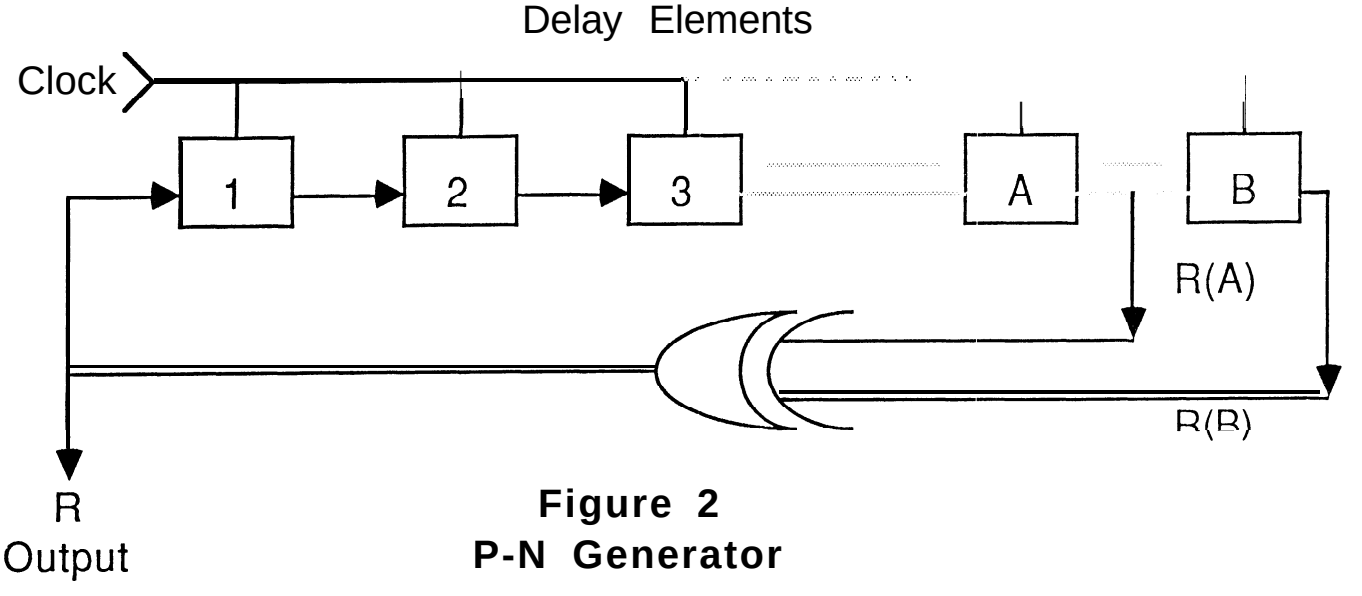


Figure 2

P-N Generator

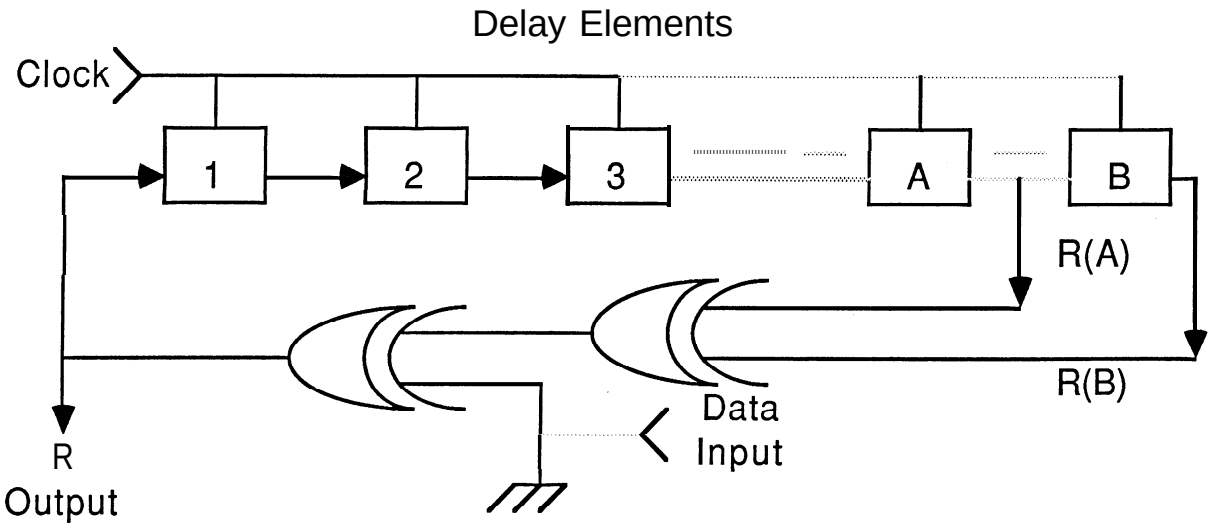


Figure 3
Randomizer

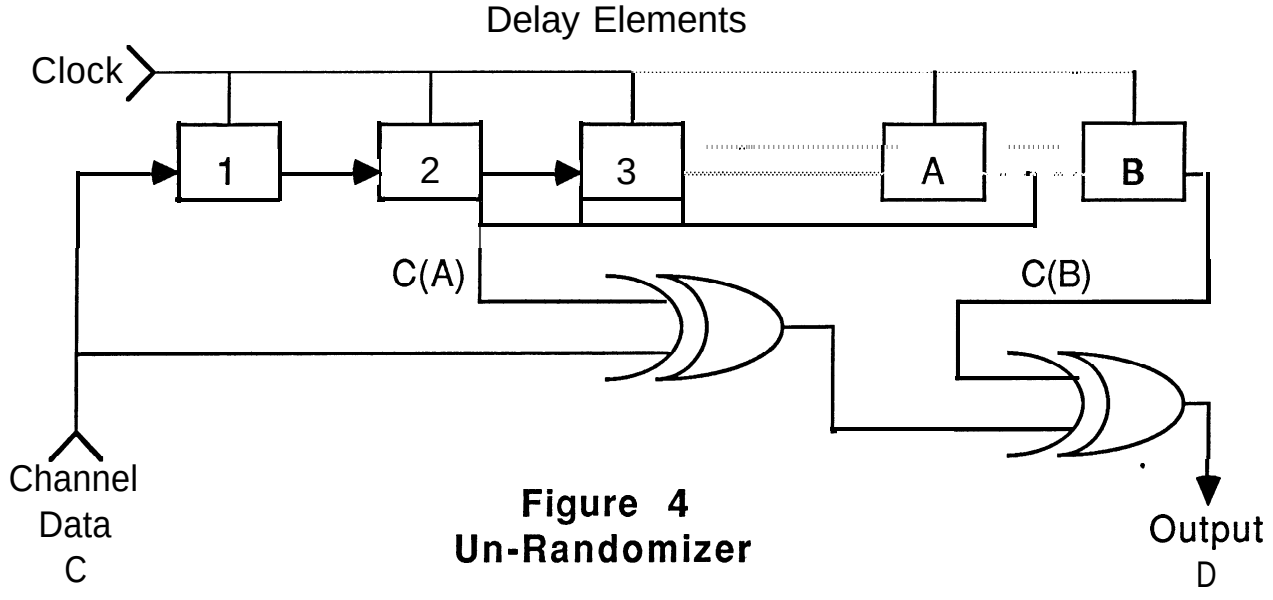


Figure 4
Un-Randomizer

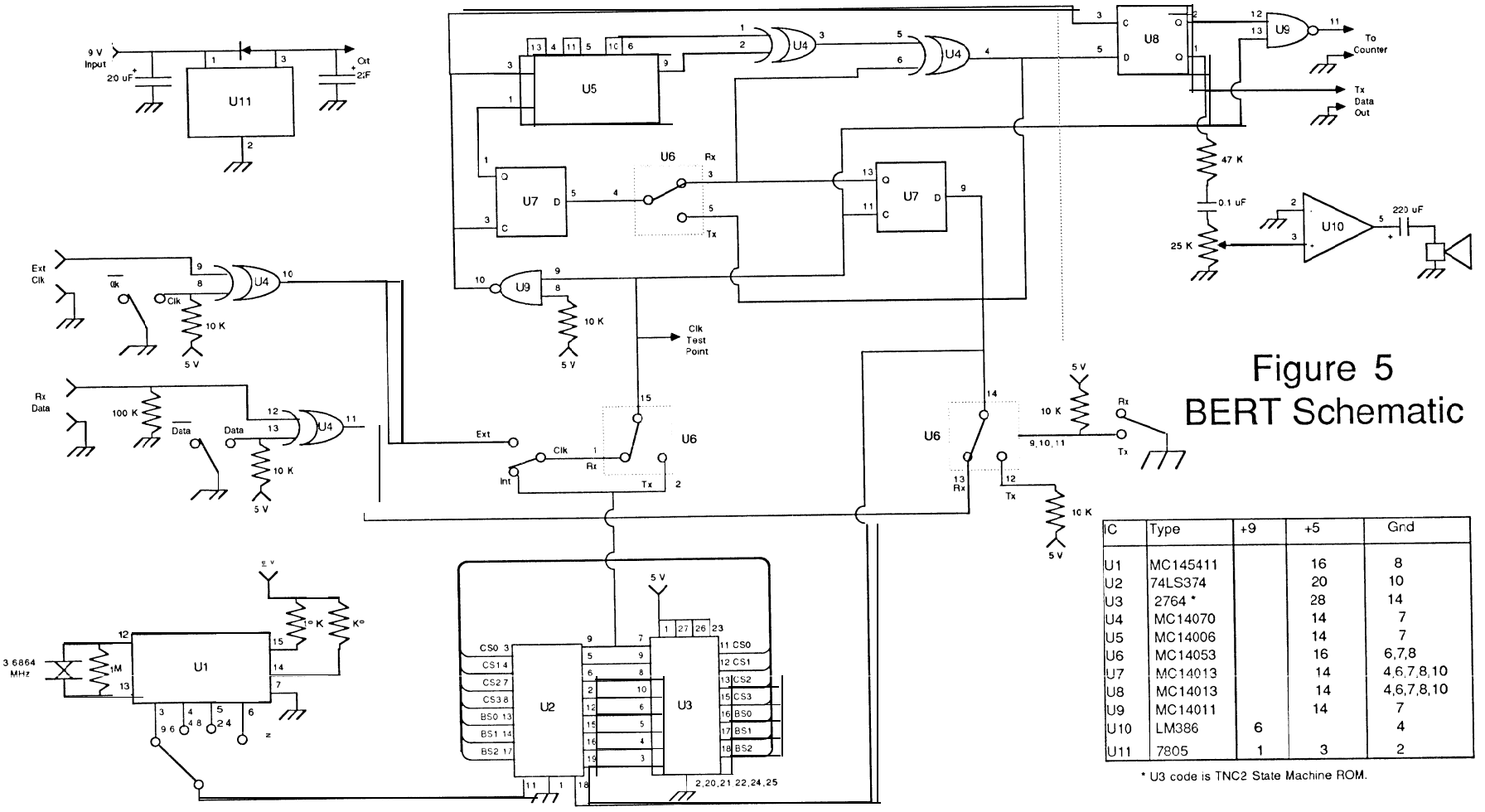


Figure 5
BERT Schematic

IC	Type	+9	+5	Gnd
U1	MC145411		16	8
U2	74LS374		20	10
U3	2764 *		28	14
U4	MC14070		14	7
U5	MC14006		14	7
U6	MC14053		16	6,7,8
U7	MC14013		14	4,6,7,8,10
U8	MC14013		14	4,6,7,8,10
U9	MC14011		14	7
U10	LM386	6		4
U11	7805	1	3	2

* U3 code is TNC2 State Machine ROM.

A 56 KILOBAUD RF MODEM

Dale A. Heatherington, WA4DSY

Abstract

This paper describes a 56 kilobaud synchronous RF modem with a 70 kHz bandwidth. The modulation is bandwidth limited MSK generated by a digital state machine driving two digital-to-analog converters, and two double balanced modulators. The carrier phase is shifted plus or minus 90 degrees for each bit. Demodulation is accomplished with a standard quadrature detector chip but various coherent methods can be used for operation at lower signal to noise ratios. The design is relatively simple and easily reproduced.

Design Philosophy and Goals

This modem was designed so that advanced Amateur radio experimenters can duplicate it. Features include:

No exotic, expensive or hard to find parts.

Stable, easy to align design.

Operation on any band allowing 56 kBaud.

Synchronous design.

Since packet radio uses a synchronous protocol, the modem was designed to be synchronous also. Characteristics of this synchronous modem are as follows:

Modem supplies transmitter clock.

Modem receiver recovers and supplies clock and data.

Modem does NRZ-NRZI conversions.

Modem scrambles data prior to transmission and descrambles it after reception.

As a result, the synchronous serial interface in the TNC doesn't need a baud rate generator, clock recovery circuit, or a NRZI to NRZ converter.

To simplify the design and construction and to allow operation on various bands, the modem operates at low power (1 mW) in the 28-30 mHz range. The final operating frequency and power output are obtained by selecting an appropriate transverter module. Good performance has been obtained with the Microwave Modules MMT 220/28s and MMT 432/28s. These transverters provide 5 to 7 watts of output power in the 220 or 432 mHz band. These units require a small modification to improve their transmit/receive switching time. A capacitor in the T/R switching circuit must be changed or removed.

Modulation

Many different modulation methods were tried in order to find one with the most desirable characteristics. BPSK was not used primarily because it has large amplitude variations (zero to full power) and uses more bandwidth than the chosen method. Amplitude variations are a problem because they must be passed unchanged by the transmitter. Any nonlinear amplifier stages will cause spreading of the spectrum and interference to adjacent channels. This effect is the same as "flat topping" in SSB transmitters.

FSK was not used because it cannot be demodulated with a coherent demodulator and it is difficult to design and build a stable frequency modulated oscillator capable of large linear frequency shifts (28 kHz).

The chosen method is a bandwidth limited form of minimum shift keying (MSK) . It is slightly different from the ordinary textbook example of minimun shift keying. Minimum shift keying is basically FSK with precise control of several parameters. The frequency shift is exactly 1/4 the baud rate and the phase of the carrier shifts exactly 90 degrees during each baud interval. The amplitude is constant. Unfortunately. MSK produces many unnecessary sidebands which make the signal very wide. When MSK is filtered with a bandpass filter to

eliminate the unwanted sidebands, the carrier phase no longer changes by exactly 90 degrees and the amplitude is no longer constant.

The method used in this design removes the unnecessary sidebands and maintains the 90 degree phase shift during each baud interval at the expense of about 3.5 dB of amplitude variation. Also, when the signal is detected using any kind of FM demodulator, the high frequency components appear to be boosted. This necessitates the use of a simple de-emphasis network following the FM demodulator. This network also reduces high frequency noise which results in improved performance.

Bandwidth limited MSK characteristics include:

26 dB bandwidth is 1.25 Hz/Baud.

Uses slightly less bandwidth than BPSK.

Error rate vs carrier to noise ratio performance comparable to BPSK when a coherent demodulator is used.

Has much less amplitude fluctuation than BPSK.

Can be demodulated with several types of detectors.

- quadrature detector
- discriminator
- differential phase detector
- Costas loop

Modulation Hardware

Modulation is accomplished with two double balanced modulator chips (type 1496). One modulator is called the "I" (in phase) modulator and the other is call the "Q" (quadrature) modulator. (See Fig. 1) . The carrier frequency is generated by a crystal oscillator operating in the 28 to 30 mHz range. The carrier drives the I modulator directly but is phase shifted by 90 degrees before driving the Q modulator. Modulation waveforms are stored in an EPROM chip. These waveforms are read out by a digital state machine into two digital to analog converters (type DAC-08). The analog waveforms are filtered with simple three pole active low pass filters to remove digital sampling noise before being sent to the I and Q modulators. The outputs of the modulators are combined, amplified, and output to -the transverter module. Signals genera-ted this way are unconditionally stable in terms of phase shift and frequency deviation.

Almost any modulation type can be generated with this hardware configuration because the digital state machine has complete control of the amplitude and phase of the carrier. The modulation characteristics are defined by the data stored in the EPROM. The same EPROM also contains the code to run the state machine and NRZ to NRZI converter.

Data rates from 1 to about 120 kilobaud are easily generated by changing the baud rate crystal and/or the divide ratio in the baud rate circuit. Six resistors in the low pass filters also need to be changed for different data rates.

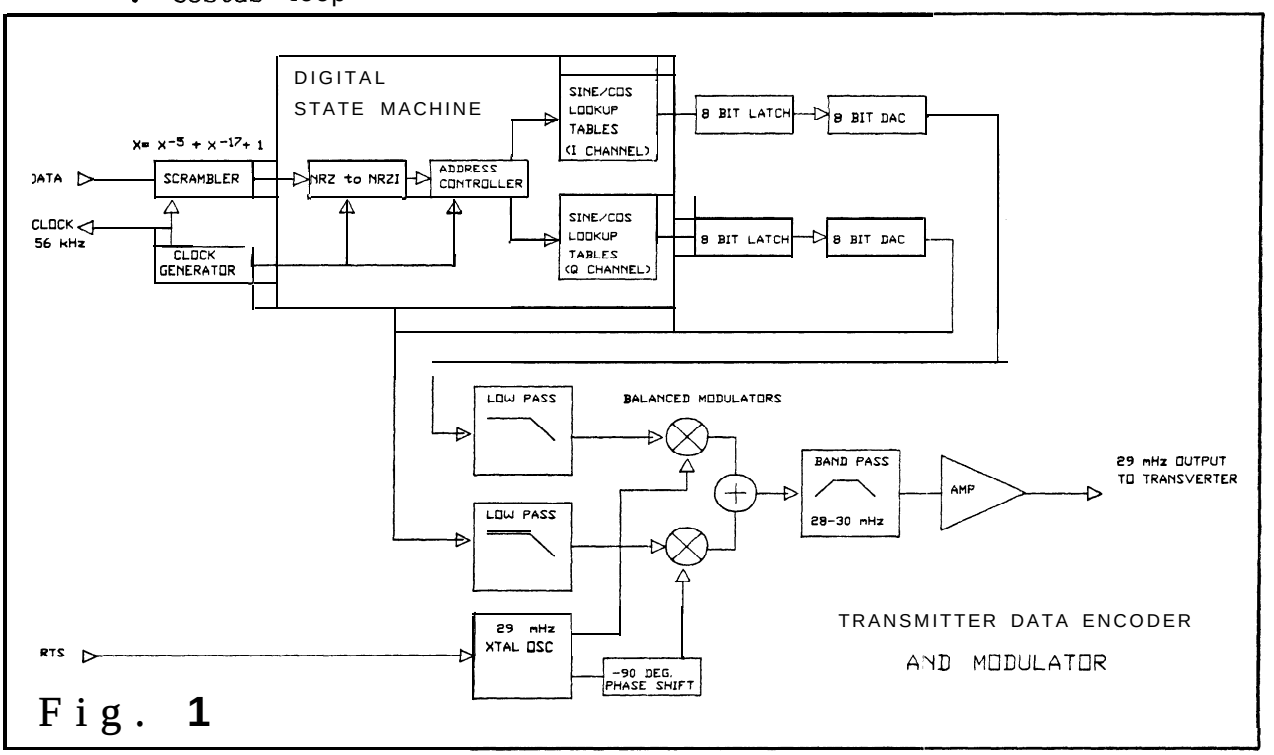


Fig. 1

Demodulation Hardware

Several methods of demodulation can be used. Since the carrier phase changes by exactly 90 degrees during each baud interval, a Costas loop demodulator could be used. The signal also has a frequency shift of 1/4 the baud rate which allows conventional FM or FSK demodulators to be used.

Costas Loop

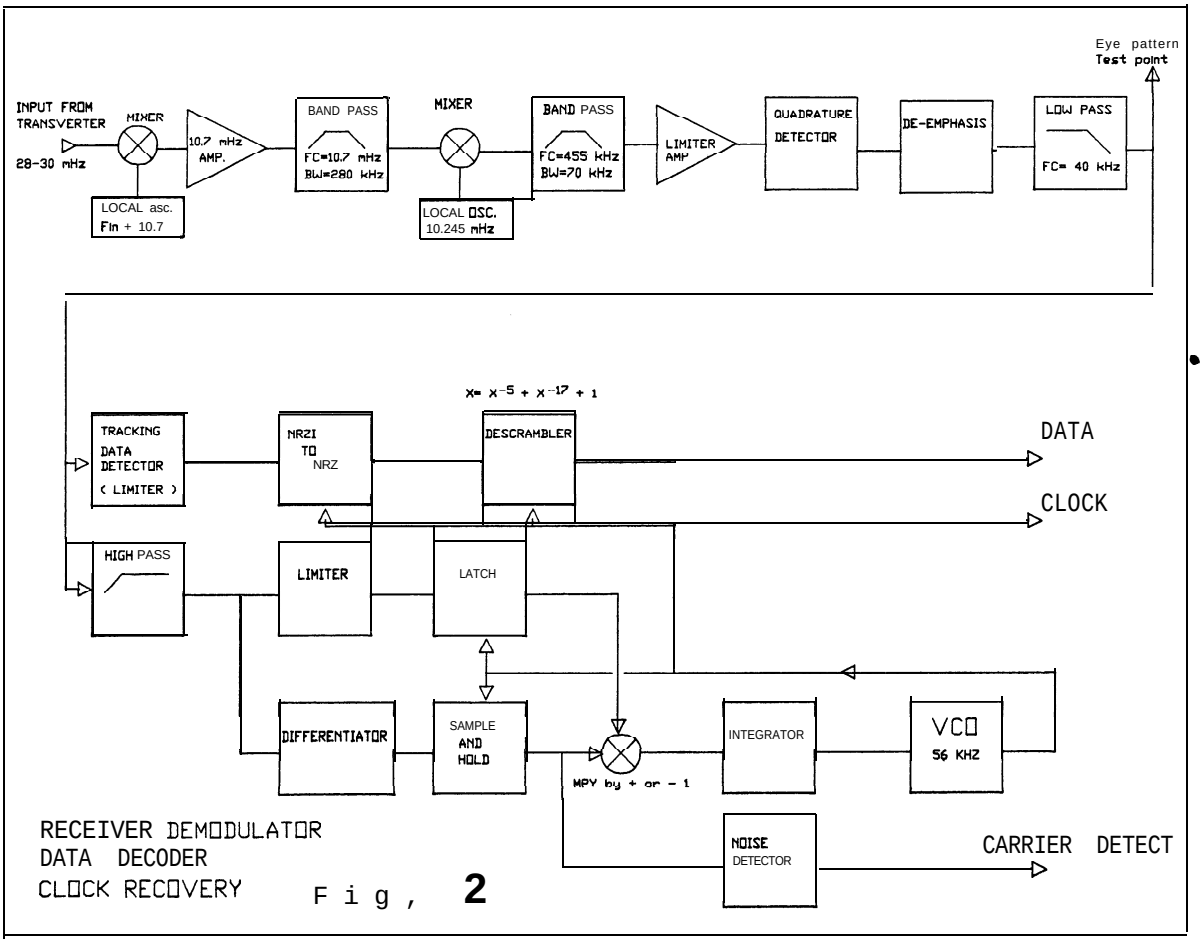
The Costas loop is an intelligent phase lock loop which locks an oscillator to the phase and frequency of the received carrier even with the presence of phase modulation. The received signal is multiplied by the locally generated carrier to recover the original modulation. The main advantage of the Costas loop is its ability to operate at low signal to noise ratios. The disadvantages are its complexity and slow lock-up time. A Costas loop demodulator was built for this project but temporarily shelved because of its 60 ms lock time and large number of parts. It did have a 5 dB S/N advantage over the quadrature detector that will be described below.

Quadrature Detector

In the interest of cost, simplicity, and fast signal acquisition, a conventional quadrature FM demodulator was used in this design (see fig. 2). The Motorola MC3359 chip was chosen for this task. It's more than just an FM demodulator. It also includes an oscillator, mixer, limiter, and several other functions which were not used.

Receiver Bandpass Filter

The receiver bandpass filter was a major stumbling block at the beginning of this project. It had to be 70 to 80 kHz wide with low group delay variations. No "off the shelf" filters could be found at any price. Custom crystal filters had very long lead times and high prices. The solution was simple and cheap: a 3 section L-C bandpass filter operating at 455 kHz. It consists of three 50 uH slug tuned coils and 5 capacitors. The cost is about $4.00. Another filter was required for the 10.7 mHz IF stages. Since its major purpose is image rejection, the response shape was not too critical. A Radio Shack FM broadcast band receiver IF filter had the desired characteristics. It's 280 kHz wide, small, and cheap ($1.00).



Fig, 2

Data and Clock recovery

After demodulation the signal must be processed to recover the clock and data. The data is detected with a tracking data detector, then converted from NRZI to NRZ format. It is then descrambled and sent to the output connector on the modem. Clock is recovered with a sampled-derivative phase locked loop circuit. This circuit aligns the active clock edges with the center of the incoming data bits.

Tracking Data Detector

A data detector is basically a analog comparator. Its threshold is set exactly halfway between the voltage level of a "1" and a "0". It outputs a "1" if the input is higher than the threshold and a "0" if it's lower. There is a problem when the carrier frequency of the incoming signal changes. The voltage levels of the ones and zeros change so the threshold is no longer exactly halfway between them. This causes an increase in errors. One common solution, which doesn't work very well, is to AC couple the output of the demodulator to the detector, This is fine if the short and long term average of the number of ones and zeros is equal. This ideal condition cannot be guaranteed even if a scrambler is used. A much better solution is to put some intelligence in the detector so that it averages the voltage level of the ones separately from the average of the zeros then subtracts the two averages to obtain the ideal threshold level. This circuit doesn't care about the ratio of ones to zeros as long as there is a reasonable number of each. A scrambler is used to make sure there is a reasonable number of both ones and zeros. The circuit will compute the correct threshold if the input signal carrier frequency is anywhere within the expected range of the ones and zeros, in this case plus or minus 14 kHz. The maximum frequency offset that can be tolerated is actually limited by the bandwidth of the receiver filter. In this implementation the error rate starts to increase slightly with frequency offsets greater than 5 kHz.

Clock Recovery

For the lowest possible error rate the clock recovery must be fast, accurate, and have low jitter. The state machine circuit used in most TNCs today has a large amount of phase jitter and is not suitable for this application. Its main advantage is low parts count (2 chips). The circuit described here is a sampled-derivative phase locked loop. It gets its phase information from the bit centers, not the zero crossings like most circuits in common use. It does this by taking the derivative (rate of change) of the demodulated data. This converts the data peaks (centers) to zero crossings. This works because the rate of change is zero at the peak of the data bit. This signal is then further processed and compared to the phase of the VCO. The result is an error voltage which is then integrated and applied to the VCO control voltage input. The VCO phase locks to the centers of the incoming bits. Lock time is about 5 milliseconds in this implementation.

Carrier Detector

The carrier detector circuit works by measuring the signal to noise ratio of the signal. This is not an easy task because the data resembles random noise in most aspects. This design solves the problem by first taking the derivative of the demodulated signal, then sampling only the zero crossings of the derivative with the active clock edge. Since the rate of change at the bit centers should be zero, the output of the circuit should be zero. Random noise has random rates of change at the sampling instants which cause the circuit to output a random voltage. This noise voltage is rectified, filtered, and sent to a comparator. If the noise voltage is below a preset threshold, the comparator turns on the carrier detect signal at the digital interface and also turns on a data gate which allows the received data to go out to the interface. An unmodulated signal will also trigger this circuit since there would be no noise at the sampling instants to inhibit the comparator.

NRZI to NWZ Converter

NRZ is a data signaling format in which zeros are represented by a certain voltage level and ones by another. NRZI is a signaling format in which zeros are represented by a change in voltage level while ones are indicated by no change. NRZI coded data is not affected inverting the data voltage levels or the mark/space frequencies in the case of FSK. This modem converts the incoming NRZI data to NRZ data with a simple circuit consisting of a "D" Flip Flop and XOR gate.

Descrambler

There are two good reasons a data scrambler was used in this modem. First, it makes the data stream look like a random stream of ones and zeros regardless of the data being transmitted. This characteristic makes the tracking data detector and clock recovery circuits work better. Second, it makes the RF spectrum look and sound like band limited white noise. In other words, the RF energy is spread evenly over the modems bandwidth and shows no single frequency lines regardless of the data being

transmitted. Any potential interference to near by channels is limited to an increase in the noise floor instead of squeaks, squawks, and other obnoxious noises. This type of scrambling is also commonly used in high speed syncronous modems for telephone use.

The hardware to implement the scrambler and descrambler is very simple It consists of a 17 bit shift register and two XOR gates. See Fig. 3. Each transmitted bit is the result of the exclusive ORing of the current data bit with the bits transmitted 5 and 17 bits times before. To descramble the data it is only necessary to exclusive OR the current received bit with the previous 5th and 17th bits. If the data consist of all ones, the scrambler will produce a pseudorandom sequence of bits that will repeat after 131,071 clock pulses or every 2.34 seconds at 56 kilobaud.

Some people may complain that scrambling violates the FCC rule concerning codes and ciphers. It does not. The scrambling algorithm is published here and is available to anyone who wants it. For this reason it does not actually obscure the meaning of the data. If all codes and ciphers were illegal we could not use Morse code, ASCII, or NRZI!

Performance

There are several measures of a modem's performance. One of the most important is its bit error rate. Any well designed modem should not produce any errors if the signal has no noise or distortion in it. The true measure of a modem's quality is its performance under weak signal conditions when the signal to noise ratio is low. These modems were tested to determine their error rate performance at various signal levels and frequency offsets. The results are summarized in the table below.

Signal level		Errors per 1 million bits Freq. offsets in kHz		
uv	dBm	0	+5	-5
.71	-110	3620	43261	11280
.79	-109	2736	8490	3110
.89	-108	843	4694	1510
1.0	-107	129	1680	285
1.1	-106	29	536	77
1.2	-105	0	240	19
1.4	-104	0	23	3
1.5	-103	0	0	0
1.7	-102	0	0	0

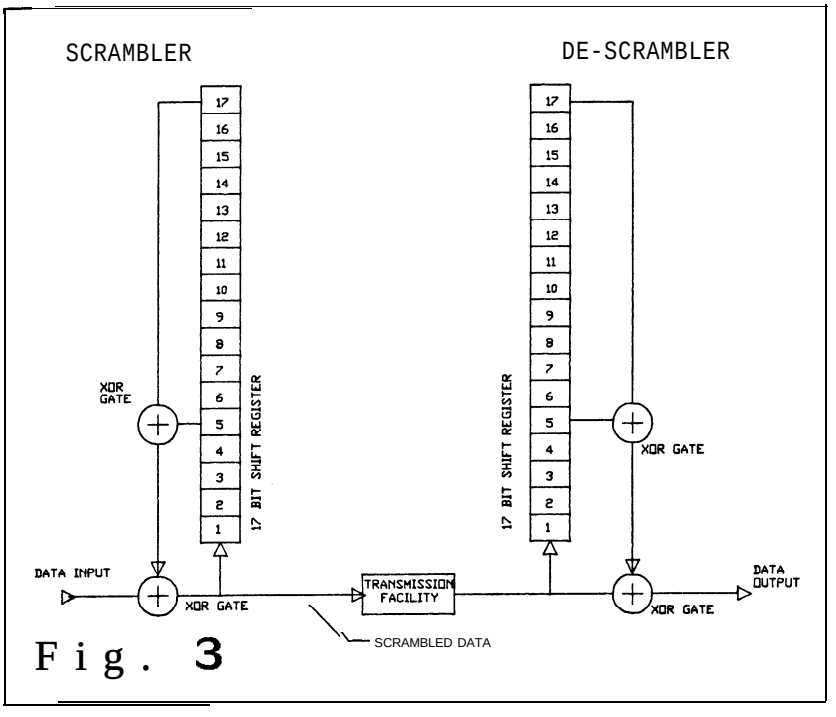


Fig. 3

The table shows that 1.5 microvolts of signal are necessary to achieve an error rate less than 1 per million over a plus or minus 5 kHz frequency range. It also shows that the performance is degraded by 2 dB if the frequency is offset by 5 kHz.

This data was obtained using a MMT 432/28 transverter without a GASFET preamp. The published noise figure for this unit is 3 dB. Adding a pre-amp should improve the performance.

Another important performance parameter is the delay time from transmitter turn-on to valid data at the receiver. This modem requires a 10 to 13 millisecond delay after the transmitter is keyed before data can be transmitted. 5 to 7 ms of this time is required for the crystal oscillator to start and stabilize. The remaining 5 to 6 ms allows the distant receiving modem time to phase lock its clock to the incoming signal and detect the carrier. This time delay is not as low as it should be and work is being done to reduce it. If the delay were zero a 256 byte packet should take 36.5 ms to transmit. This modem takes up to 50 ms of transmission time or 36% overhead. This overhead percentage could be reduced by transmitting longer packets.

Applications

So, what do you do with a 56,000 baud RF modem? One obvious application which has received much press lately is network backbones. One popular opinion seems to be that high speed modems will solve network congestion problems. This assumes that current firmware and software will run at 56 kilobaud. It doesn't! One of the major problems of this project was getting packet software to operate at this speed. No TNC running AX.25 will operate at 56 kilobaud. IBM PCS running at 8 mHz can only handle serial port data at 38.4 kilobaud or less without dropping characters. To conduct on the air tests it was necessary make major modifications to a Z-80 assembly language program called KISS-TNC written by K3MC. This program resides in an EPROM in the TNC. Its main job is to convert SYNC frames to ASYNC frames and send them to the serial port on a PC. A program running in the PC is responsible for doing the protocol, in this case NET.EXE by KA9Q, which implements TCP/IP. Although KISS-TNC was successfully modified for 56 kilobaud, NET.EXE would not run any faster than 19.2 kilobaud. The result was that the PC to TNC interface ran at 19,200 baud while the TNC to RF modem interface ran at 56,000 baud. Needless to say, NET.EXE could not keep the channel busy. This may not be a problem because the channel is a shared resource and should not be hogged by one user anyway.

Digital Voice

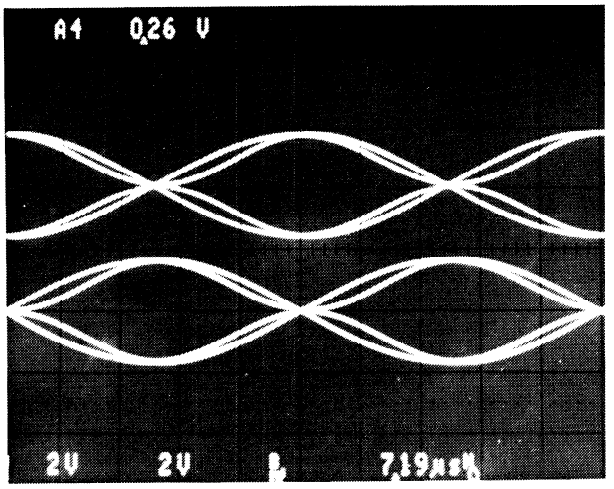
This modem is fast enough to carry real time digitized speech. Two methods of converting speech to digital format are popular. The phone company uses Pulse Code Modulation (PCM) at a 64 kilobaud data rate. Unfortunatly someone set the upper limit for amateur digital communication at only 56 kilobaud thus preventing the use of many cheap CODEC chips in the market today. They only work at 64 KBPS. One good alterative to PCM is continuously variable slope delta modulation (CVSD). Distorted but intelligible speech can be transmitted at speeds as low as 9600 BPS with CVSD. At 56 KBPS it sounds as good as typical communications quality FM. The CVSD chip used in the experiments with this modem is the Motorola MC34115. Its price is in the $2.00 range. CVSD also works well in the presence of data errors. Digitized speech can be understood with more than 10% bit errors. It's quite noisy at that error rate and sounds like a weak FM signal.

Digital Video

Video frame grabbers are available which digitize and store pictures from a NTSC video source in computer memory. Images stored in this manner can be transmitted digitally. A video frame consisting of 256 by 256 pixels with 64 shades of gray can be transmitted in 7 seconds at 56 kilobaud. The same image would take more than 5 minutes to send with the current 1200 baud standard.

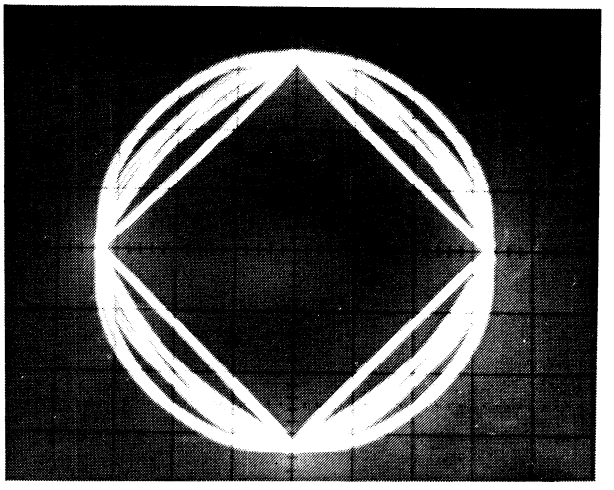
Conclusion

Bandwidth limited MSK is a power efficient modulation method with reasonable bandwidth requirements and low amplitude fluctuations. The hardware required for generation and demodulation is simple and reliable. A wide variety of demodulators can be used depending on the cost/performance tradeoffs. It is an excellent choice for high speed RF data links.



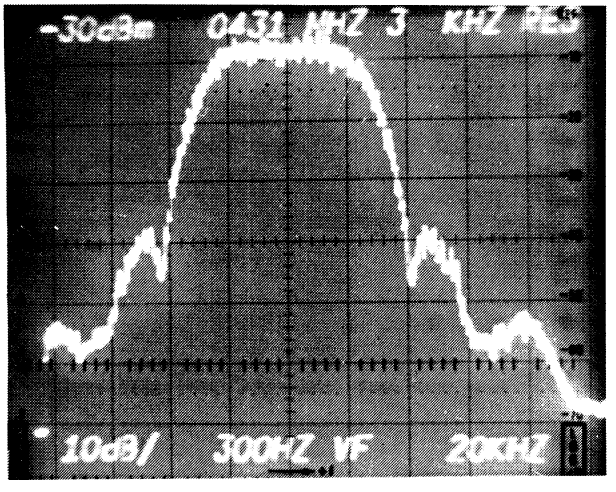
Modulation waveforms

Top trace : I modulation
lower trace : Q modulation

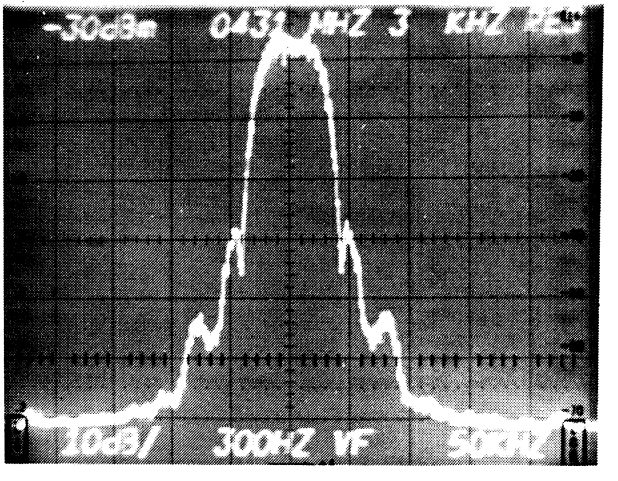


Signal constellation

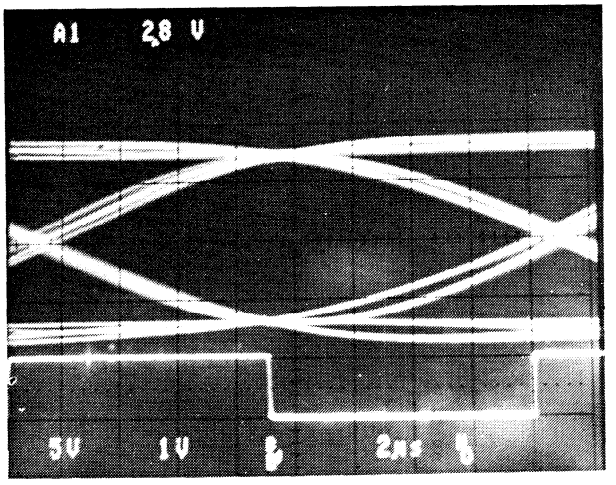
Transmitter X,Y display
X axis : I modulation
Y axis : Q modulation



Bandwidth limited MSK spectrum
20 kHz/div. horz.
10 dB/div. vert.

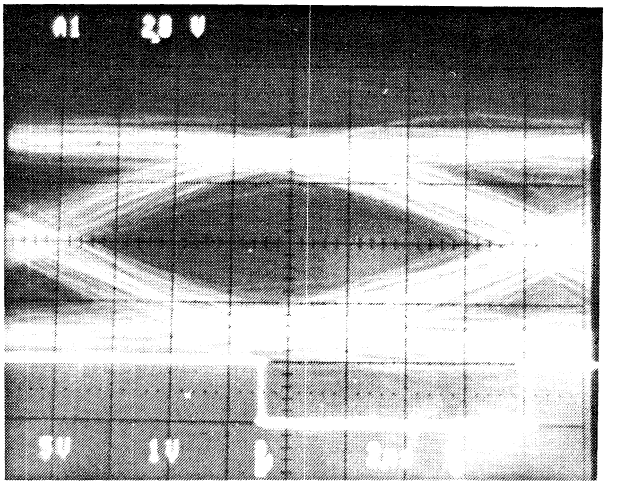


Bandwidth limited MSK spectrum
50 kHz/div. horz.
10 dB/div. vert.



56 kilobaud EYE

Top trace : Receiver EYE pattern
Lower trace : recovered clock



56 kilobaud EYE

Top trace : weak signal EYE pattern
RF signal level is about 1.4 uV
lower trace : recovered clock

Reusable IP Addresses in a Dynamic Network

Robert B. Hoffman, N3CVL

ABSTRACT

The topology of amateur packet radio networks changes rapidly due to the frequent addition of new stations, shutting down of old stations, and changing location of others. This paper presents a method for managing IP address assignments within such a network.

1. Background

TCP/IP networks require that each host have a unique S-bit address. These addresses are typically assigned by the network manager who must make sure that no duplicate addresses exist. In the amateur packet radio TCP/IP network, the assignments are done in a hierarchal fashion. The global coordinator (GC) assigns blocks of addresses to Local Area Network (LAN) coordinators who, in turn, assign individual station addresses.

The amateur packet radio community is constantly changing due to the adding of new stations, shutting down old stations, changing locations, and the like. In the AX.25 digipeater network, it becomes difficult to maintain an accurate map of reliable connection paths. In the TCP/IP network, the job of the LAN coordinator becomes similarly difficult

When a new station comes on the air in the TCP/IP network, its operator must first contact the LAN coordinator to get an address assignment If the coordinator is unavailable, the new user may get frustrated and choose a random address which may confl ict with previously assigned addresses, causing havoc on the network. In order to ease the adding of new stations to the TCP/IP network, the process of address assignment must be automated.

2. Automatic Address Assignment

It can be assumed that the LAN coordinator operates the router for his LAN and that it has knowledge of all LAN address assignments. It therefore has enough information to be able to assign addresses within the block assigned to it by the CC. When a new station comes on the air, it sends a broadcast packet that contains its callsign and a request for a "permanent" IP address. The LAN router searches its tables for the station's callsign, and if it is found, it responds with the previously assigned address. If a table entry is not found, the router allocates a new address from its block and assigns it to the requesting station. It also makes an entry in its tables linking the station's callsign with that address. This is similar to the Reverse Address Resolution Protocol [1] that is used in booting diskless workstations. The router then sends a packet to the requesting station informing it of its assignment The requesting station then records the assignment in its configuration fi le for subsequent use.

When the current block of addresses is exhausted, a new block would have to be requested from the CC. Currently, the LAN coordinator must make a request to the GC for another block of addresses. As the network develops better connectivity, we may be able to have the LAN router send a special packet to the CC's system to request another block of addresses. The CC would take the next available block and mark it as being assigned to that LAN, and send the information back to the originating LAN router. At the same time, the new block-to-LAN assignment is distributed to all other routers so that they may update their tables. The LAN router may elect to send its request when a few addresses are still unassigned in the old block, to allow for delays in response from the CC.

The LAN will also have a name server which will probably operate on the same system as the router. Its function is to accept packets contain-

ing callsigns and return the associated IP addresses.

3. Address Expiration

The local IP assignments may have an expiration date associated with them so that seldom-seen stations don't tie up IP addresses needlessly. This can be an arbitrarily long time, such as a couple of months. As long as a station remains active at least once during that time period, it retains its assignment and stays in the name servers. If an address expires, it is marked as being available for the next new station. This will lengthen the time before a new address block is needed.

4. Moving between LANs

When a station moves from one LAN to another, its IP address would be marked as invalid in the local router, and made to point into a forwarding table that indicates the station's new IP address. This would be maintained for some time to insure that the new IP address has had time to show up on the network's name servers, and so that the old address does not get reassigned locally until a reasonable time has passed. The rules that govern routing decisions that are made based on a partial IP (subnet) address cannot allow IP addresses to move between LANs. This is necessary because one cannot unplug a computer from one organization's network and relocate it to another organization's network and expect to keep the same IP address. With domain style addressing, it wouldn't even have the same hostname.

5. Mobile Stations

For mobile packet stations operating away from their home territory, a temporary address would be requested from the router in the station's current LAN. The local router then sends a forwarding order to his "home" router, cancelling any previous forwarding order. The home router then sends a cancellation order to the previous router so that the previous temporary address may be purged. The temporary address would have a much shorter expiration time than a regular address. This scheme assumes connectivity between all of the LANs on the mobile station's route.

6. Conclusion

As the number of stations using TCP/IP grows, it will become increasingly important to respond quickly to changes in the network. For this reason, some sort of automated network manangement is necessary. The ideas presented here represent a method for managing IP address assignments in such a network.

References

a. Finlayson, R., Mann, T., Mogul, J., and Theimer, M., "Reverse Address Resolution Protocol," ARPA RFC 903, June 1984.

Acknowledgements

I wish to thank Mike Chepponis, K3MC, and Bdale Garbee, N3EUA, for their assistance and encouragement in the preparation of this paper.

Software Design Issues for the PS-186 Advanced Packet Network Controller

Brian Kantor, *WB6CYT*
Academic Network Operations Group
Office of Academic Computing
University of California, San Diego

Abstract

A fast network for amateur radio requires sophisticated node controllers to work well. Key to the performance of advanced node controller hardware is the design of the on-board software. Issues of highly-efficient device drivers, protocol encapsulation, and process management must be addressed to ensure acceptable performance with limited memory and affordable hardware. PS-186 hardware design issues are discussed in a companion paper by Michael Brock, Franklin Antonio, and Tom LaFleur.

1. Introduction

A high-throughput data network must consist of both high speed links and fast network node controllers. To achieve the high throughput in the controller requires both good hardware and efficient software.

The PS-186 offers a highly efficient hardware design including very high speed input/output and a fast processor. The PS-186's high-speed DMA channels allow much of the I/O to proceed in parallel with computation, thus overlapping I/O and processing that in a less sophisticated system might need to proceed serially.

However, even the most advanced hardware can be crippled by inefficient software that wastes CPU and I/O resources rather than applying them to useful processing. To take full advantage of the PS-186 architecture, we have chosen to use a multi-tasking system that can support several programs running at once. By dividing up tasks into those that are time-critical and those that are not, we can set up the critical tasks in the system such that they will receive the required CPU attention. Less critical tasks will proceed as time permits.

The PS-186 multi-tasking operating system is based in general terms upon the **UNIX**@ and other similar simple operating systems. In particular, many of the ideas and practices used have been taken from those presented in the MINIX [IO] and XINU [2,3] model operating systems.

2. Software Organization

The PS-186 operating software can be divided into two categories. One of these is the central or core part of the system, referred to as the *kernel.* It is responsible for all supervisory low-level I/O functions, process and memory management, interrupt handling, and time-critical tasks. The remaining software is termed the *user* **level** software, although there are, strictly speaking, no users on **this** system. The true distinction is that while there **may** be many "user" processes running at one time, there is only one kernel.

Author's current address:
electronic: brian@sdcsvax.ucsd.edu
paper: Office of Academic Computing B-028, La Jolla, CA 92093 **USA**

User processes can be stopped while they are waiting for input, or while the kernel is handling some other event such as an arriving packet. They are typically used for purposes that are not time-critical and that can operate independently of particular hardware status. A few examples of tasks that might better be placed in user processes are table lookups, help menu displays, and the like. These are things that can proceed in parallel with other similar tasks.

The kernel, on the other hand, is strictly *single-threaded* – it can only execute by itself, and is intended for those tasks that need to have exclusive access to the processor or devices, such as interrupt handlers and device drivers.

User processes do NOT directly access devices, nor do they handle interrupts. All user processes communicate with the kernel by means of *system calls* that cause the kernel to perform some task on behalf of the user process. A common example is a read or a write – data transfer between a user process and a device.

The kernel is responsible for performing all encapsulating protocols below some arbitrary level, which we have chosen to be at the "data stream" level. That means that when an AX.25 connection is made to the PS-186, the kernel software is responsible for the acceptance, acknowledgement, and eventual knockdown of the connection. The kernel will extract the data field from the incoming AX.25 packet and make it available to a user process executing an appropriate *read.* Likewise, a user process that wishes to send data over an open AX.25 connection will give the data to the kernel by means of a *write* system call, and the kernel will do the encapsulation necessary to send the data via AX.25 and then queue it for transmittal by the appropriate device.

When there are multiple protocols involved, such as TCP-IP on AX.25, the kernel does the multiple extractions and encapsulations as required, so that the user process again works only at the data level.

The decision on whether to place a particular task in the kernel or leave it to user-level processing is based on a number of criteria, some of them empirical. In general, any task which can wait to complete without impacting the performance of other tasks can generally be placed in a user-level process, whereas tasks that have a number of process dependencies pretty much have to go inside the kernel. Additionally, any protocol encapsulation or unwrapping that does not generate additional packets can be placed in the kernel, thereby making that function available to all user-level processes by means of a uniform system call.

3. Input-Output

Each device in the PS-186 has a module of code associated with it in the kernel that does the low-level input-output interface to the actual hardware. This module is often referred to in the literature as a *device driver.*

At the low level hardware interface, a device driver is responsible for taking data to be output from some generalized system data structure, and actually outputting it through the corresponding piece of hardware. It also must accept incoming data from the hardware device and place it into a system data structure for further processing. It is common for device drivers to operate in an *interrupt-driven* mode, with their actions being invoked in response to "completion" or "ready" signals from the hardware. We have chosen this method over a perhaps simpler scheme where the main software loop simply repetitively checks for device availability, because the latter scheme potentially wastes a tremendous portion of the available processing power. Additionally, the various drivers make use of the PS-186's DMA (Direct Memory Access) capability to move the actual data between memory and the device without the need of the CPU to read and write every byte.

There also must be a simple and consistent higher-level interface to the system data structures that the low-level device drivers access. We have chosen to implement this interface as *read* and *write* system calls that invoke high-level portions of the device drivers. Additionally, there are both high- and low-level configuration, status, and initialization functions that are logically part of the device driver. Thus each driver can be divided into two logical functions, referred to as the "top" and "bottom" of the driver.

The "bottom" function is the interface to the hardware; it is invoked in response to an interrupt from the actual device. Typically its sole function is to move data to and from the device and an associated memory buffer or buffer queue.

The "top" function is the interface to the kernel *read* and *write* system calls. It does the opposite of its corresponding "bottom" half; where the bottom half places incoming hardware data into a buffer, the top half will remove the data from the buffer and give it to the process executing the *read* kernel call.

When the PS-186 kernel has data to be sent out of a serial port (in response to a *write* system call), the device driver is called to accept the outgoing data. The driver adds the data to the tail end of a queue of data waiting to be sent, sets a flag indicating that there is indeed data to be sent, and returns. Later, when the output device finishes with the data it was sending, it will cause an interrupt to occur, and the "bottom half" of the appropriate device driver will take the next chunk of data from the queue and send it to the device. Thus the kernel (and therefore user processes) need not wait for I/O on a device to complete before resuming proceeding.

Data buffers and queues are dynamically allocated; when data is received or generated a "buffer" (a block of memory) is allocated from the pool of available memory to hold it, and a "pointer" that contains the memory address of the block is set up. To save the time that would be wasted in copying from one block of memory to another, data is passed from module to module by passing the pointer to the memory buffer in which the data resides, rather than copying the data itself. When the data is finally consumed, either by being output by a device, or copied into a user-level (outside the kernel) buffer by a *read* system call, the memory space used is returned to the available memory pool, and the buffer pointer is dereferenced.

When a call to the kernel with data to be output (a *write* call) would require allocation of more memory buffer space than is allowed, the process making the call is stopped by the simply not returning from the system call to that process until there is space and the write can be completed. Since "blocking" the process in this manner does NOT stop interrupt service nor other kernel functions the device will eventually output enough data to free up sufficient memory for the write to complete and for the user process to resume.

On input, if a chunk of data arrives and there is no memory available to hold it, the only practical procedure is to simply discard the data. We anticipate having a large amount of memory available for data buffers, as well as expecting good throughput, so we do not anticipate that it will necessary to discard data often. As a practical note, we have decided to provide each input device with its own memory buffer limit so that no one device could hog all available memory and shut out input from other devices even in the most pathalogical of cases. The overriding assumption is that higher-level protocols will handle packets lost due to memory congestion in much the same way that packets lost due to collisions or channel congestion are handled.

A kernel *read* call will return data from the input queue to the user process; if there is no data in the queue, the user process may elect to wait until there is ("read-wait") or just return ("read-no-wait"), When data is available, it is copied into a buffer space provided by the user process (typically a character array), and the memory buffer space is released to be reused on subsequent input events.

One can view the input-output streams as a series of filtered interfaces to the raw packets that are being received or sent. Thus it is possible to open a connection that consists of raw AX.25 frames, an AX.25 connected mode stream, IP packets in SLIP, IP packets in AX.25, TCP in IP in AX.25, etc. This is controlled by the parameters passed to the kernel in the open system call.

4. Devices

The PS-186 devices that are most interesting are the several serial controller chips that form the communications interfaces. (There is an SCSI controller option for general device access, such as to a disk or floppy controller, but we will not discuss that here.) The serial controller chosen was the Zilog 8530 SCC; the hardware design considerations that lead to it being chosen are discussed in the companion paper on the hardware design of the PS-186.

The 8530 SCC can do both asynchronous serial I/O (as perhaps to a terminal or printer), and HDLC synchronous, such as is used in the AX.25 protocol. Any of the PS-186's serial ports can be configured to operate in either of these modes. We therefore have a more complex device driver than if the PS-186 had fixed serial port allocations, since the device driver must be able to handle both sync- and async-configured devices based on parameters stored in a table: The driver is also responsible for setting up the modes of the serial ports in the first place.

5. Protocol Handling

Fundamental to the operation of communications protocols is the concept of *layering* or *encapsulation*, whereby data is successively encapsulated or "wrapped" in layers of protocol as it is prepared for transmission, and "unwrapped" at its destination.

The basic concept used is known as a switch. As a railroad switch controls the path of a train, the switch controls the path that data takes through the various levels of encapsulation and unwrapping. A protocol *switch* makes the data passing decision based on a field contained in each protocol's header that indicates what kind of protocol may be further encapsulated within the data field of the current packet.

The protocol handling scheme that we chose to use in the PS-186 is located in the kernel software. By keeping all protocol wrapping and unwrapping inside the main single-

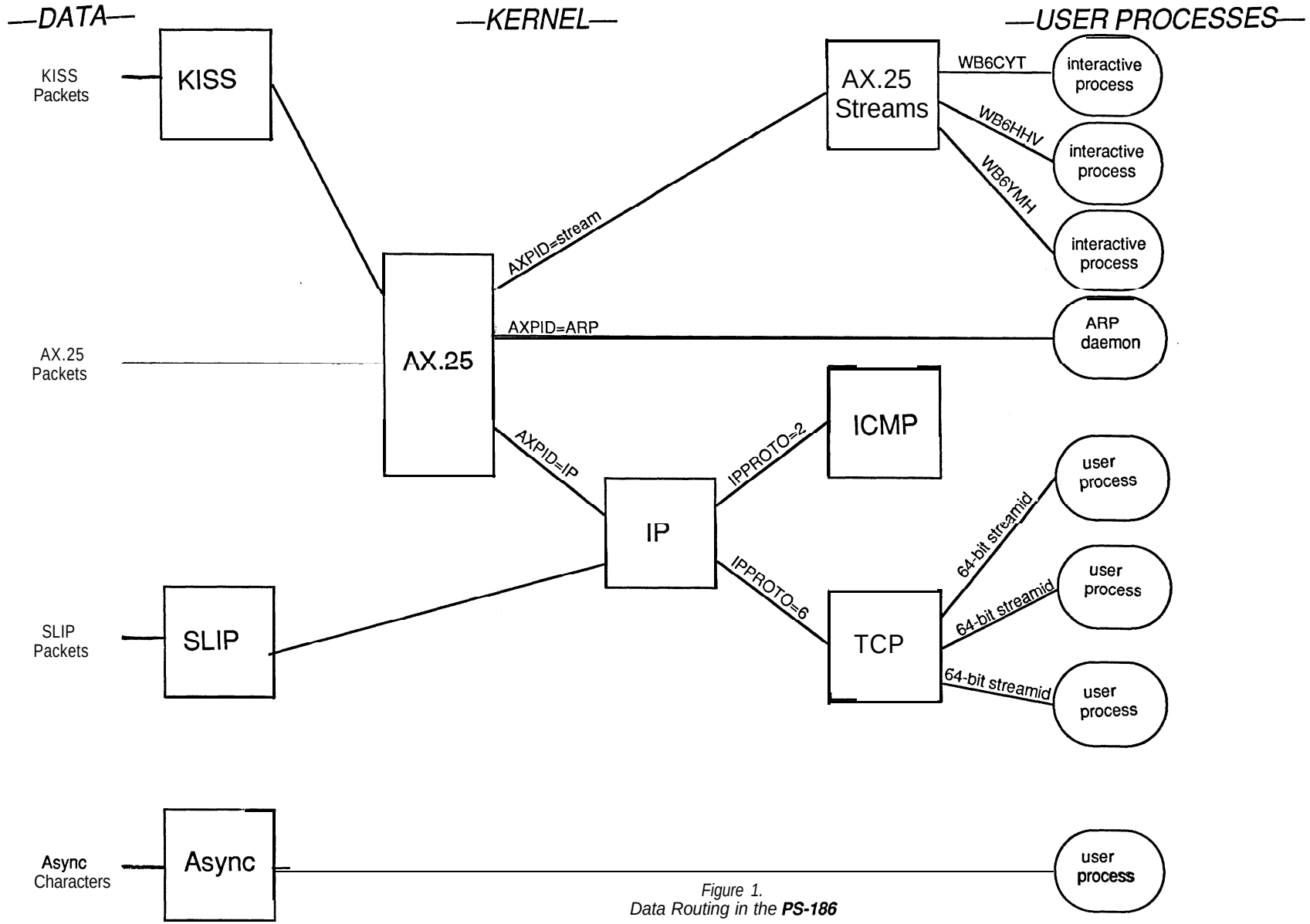


Figure 1.
Data Routing in the ***PS-186***

Showes the routes that data packets may take depending on protocols involved. Data is received as packets at left; as it passes through the various protocol handlers and switches it is sent to other protocol modules as required until finally it is fully unwrapped and available to the user processes at the right. Data to be sent follows essentially the reverse path from the user process through successive layers of encapsulation until it is ready to be sent. The criteria for determining further processing are shown on the lines connecting the various boxes.

thread portion of the operating system and thus making them available to all processes on the system, we simplify greatly the amount of protocol handling required in the various other processes.

Each PS-186 communications interface is configured at system startup time to handle one type of outermost protocol (for example, KISS AX.25 is appropriate for a serial interface to a radio link, or perhaps SLIP for a hardwire line.) As a packet is received from an interface, it is examined according to the rules that have been set up for that interface. When that packet has been received and the appropriate acknowledgements generated, the contents of the packet and selected fields extracted from its header are passed to the appropriate protocol switch. The protocol switch then examines the packet contents and routes the data further to the next protocol module, as appropriate. This process repeats until there is no further enclosed protocol, until the data has been fully extracted and is available in a buffer queue to be used by some user process. Not until the data has been fully extracted does it become ready to leave the kernel environment.

A concrete example may make this clearer: Suppose that we receive an AX.25 packet on a serial link that is attached to a radio. If it is for us (as shown by the destination callsign) we will acknowledge that AX.25 packet (if appropriate), and if it was a data packet (UI or I frame), we will pass it to the AX.25 protocol switch. That switch will examine the Protocol ID byte that is part of the AX.25 packet. If the PID is for a stream connection (a normal mode AX.25 connection such as is commonly in use today for keyboard-to-keyboard typing), then the packet will be further switched by the AX.25 Stream switch, which will send it (based on the callsign in the source field of the AX.25 header, since there can be only one connection per source callsign) to the user process that is servicing that stream connection.

If, instead, the PID is for ARP, RARP, RIP, or one of the other raw packet protocols, the data will be sent to the user process that handles that kind of packet - to build address or routing tables, for example.

A packet with the PID indicating an encapsulated IP packet is passed to the module that does IP protocol - checksums and other integrity checks. If the packet is ok by IP standards, the IP module will call the IP protocol switch, which will examine the Protocol ID byte in the IP packet (distinct from the PID in the AX.25 packet). This will in turn route the IP packet to another protocol handler, such as UDP, ICMP, RDP, or TCP - whichever we have implemented. Again, those are expected to route the data based on fields in the headers of these protocols.

TCP is an interesting example of imbedded protocols and switching. Each TCP connection as seen on a host is designated uniquely by a 64-bit number that is the concatenation of the distant host's Internet address (32 bits) and the local and distant TCP logical port numbers (16 bits each). Since when a TCP connection is initiated, the originating host must chose a new (not currently nor recently used) logical source port number, there can be multiple logical connections between, TCPs on the same two hosts even to the same distant port. The data switch in the receiving TCP is required to separate out the streams of data based upon the 64-bit stream identifier, and deliver each to potentially separate user processes as appropriate.

6. Process Control

The PS-186 is organized as a multi-tasking System, implying that more than one process may be running at a time. There is always a "null" process that is constantly ready to run; when there is no other process ready to run, the null process is active.

Processes are created as needed and destroyed when no longer needed. In this manner, resources are not consumed on idling processes that are merely sitting around waiting in case they are needed. For example, when an AX.25 connection is made to the PS-186 network node, a process is started to handle incoming stream data. This process will exit and its resources will be deallocated when the connection is closed. Each such connection will cause a separate process to be spawned.

User-level processes do not perform I/O operations to devices; they instead make *system* calls to the kernel that invoke the required I/O. When a process makes a system call that would require some time to complete (such as I/O), that process is *blocked* - that is, placed in suspended state, and another process is resumed. Periodically (in response to interrupts from the system real-tirne clock), the current process will be suspended and another selected to run. Thus no process can hog CPU resources, and I/O can proceed in parallel with ordinary processing.

We feel that multitasking is a superior method in this application, although it is much more complex than a single-threaded program, because much of what goes on in a device like the PS-186 is not time-critical, and we can therefore devote the CPU to high-priority events (such as the arrival and buffering of a packet) that are truly critical.

7. Conclusion

We feel that the PS-186 Advanced Packet Controller represents a significant step towards the construction of an efficient and practical amateur radio data network. By combining fast hardware and efficient software into a flexible package that accomodate today's and tomorrow's protocols, we believe we have advanced the network one step further along the road to completion.

8. References

[1] AT&T, "Communications Protocol Specification BX.25", Publication 54001 Issue 2 (June 1980)

[2] Comer, D., *Operating System Design – the XINU Approach,* Prentice-Hall (1984)

[3] Comer, D., *Operating System Design – Internetworking with XINU,* Prentice-Hall (1987)

[4] DEC/Intel/Xerox, "The Ethernet – A Local Area Network Data Link Layer and Physical Layer Specification.", Version 1 .O (Sep 30 1980)

[5] Fox, T. L., "AX.25 Amateur Packet-Radio Link-Layer Protocol", Version 2.0, ARRL (Oct 1984)

[6] Griffiths, Georgia, and G. Carlyle Stones, "The Tea-Leaf Reader Algorithm: An Efficient Implementation of CRC-1 6 and CRC-32", Communications of the ACM, 30,7 (July 1987)

[7] IEEE, *Logical Link Control,* ANSI/IEEE Std 802.2-1985 (1984)

[8] Postel, J. et al., "DDN Protocol Handbook", USC-ISI (1986)

[9] Tanenbaum, A., *Computer Networks,* Prentice-Hall (1981)

[10] Tanenbaum, A., *Operating Systems – Design and Implementation,* Prentice-Hall (1987)

Another Look at Authentication

Phil Karn, KA9Q

ABSTRACT

A simple and effective technique for packet authentication in a datagram network is described that is based on the Data Encryption Standard (DES). In accordance with FCC rules, the actual data is not encrypted; rather DES is used to compute a special "cipher checksum" that is appended to the unencrypted user data before transmission. The recipient recomputes the cipher checksum and compares it against the incoming value, thereby detecting bogus or altered packets. This technique is potentially useful in a wide variety of amateur radio applications in addition to packet radio; for example, it could provide a secure control link for a remote repeater site.

1. Introduction

At the 5th ARRL Computer Networking Conference, Hal Feinstein described a technique for authenticating amateur packet transmissions [1] useful for protecting radio control links. In the commercial and military worlds, the solution is simple: just encrypt everything. Not only does this protect against unauthorized commands from those not knowing the key, it also hides the command information itself. In the amateur service, however, this is prohibited by FCC rules. [2] The only exception is for command links in the Amateur Satellite Service; [3] there is no corresponding exception for terrestrial links.

Fortunately, as Hal showed in his paper, a rule change is not necessary to add security legally to a control link. Cryptographic techniques can be used to add "authentication" to an unencrypted command to verify that it was sent by an authorized station. Hal assumed that a virtual circuit would be set up between the control station and the remote site, and as a result his technique is quite involved. I have devised a similar but much simpler approach made possible by the use of a datagram protocol.

2. The Data Encryption Standard

As in Hal's approach, I use the Data Encryption Standard (DES). [4,5] A full description is outside the scope of this paper; however, I will review two of its properties since they are important to the proposed authentication scheme.

1. The internals of DES are public knowledge, since the complete specifications have been widely published. Like a well-designed safe, however, knowing how DES works isn't much help in cracking it; only the key (or the combination) need be secret for good security. This is what allowed DES to become the first-ever cryptographic standard; similarly it allows us to establish an amateur authentication standard so that each system operator doesn't have to reimplement the wheel.
2. In practice, DES has been highly resistant to "known plaintext" attacks. [6,7,8,9,10] That is, even with a plaintext/ciphertext "matched pair", the algorithm is so nonlinear that at present there is no known way (outside of the NSA, at least) to find the key that produced the transformation other than by trying all possible 2^{56} keys in the algorithm until you find the one that works. As we will see, this property is very important to the authentication scheme described here, since each transmission contains both the ciphertext and the plaintext that produced it. It

should be pointed out, however, that as yet there is no published mathematical *proof* that a known-plaintext attack against DES requires an exhaustive search. Despite some "suspicious" structure in the algorithm that reduces the required brute-force effort somewhat, the accusation that a "trap door" may have been planted in the algorithm by its designers has yet to be either proven or disproven.

As an aside, it is this resistance to known plaintext attack that makes the M/A-Com Videocipher system (which encrypts the audio with DES) so hard to break. Anyone can buy a box, subscribe to a service and get millions of ciphertext/plaintext pairs. However, that doesn't help in finding out the DES key in use, which you would need to build a pirate decoder. The key is kept inside the decryption device in a battery-backed register which can't be read from the device pins. Only a physical attack is likely to work here, e.g., dissolving the epoxy off the chip with solvents and reading the key from the exposed chip with a scanning electron microscope of the type designed for debugging ICs under test. This illustrates an important practical point in cryptography: key security, not the mathematical strength of the encryption algorithm, is usually the weak link. Key security is harder than you might think, especially when many people are involved.

3. DES Modes of Operation

The basic DES algorithm transforms 64-bit data blocks between plaintext and ciphertext form under control of a 56-bit key. As long as the key is constant, enciphering a given 64-bit data block always gives the same 64-bit block of ciphertext. If you encrypt data directly in this way, you are using DES in the Electronic Code Book (ECB) mode. ECB can result in repeated patterns when, for example, strings of blanks are encrypted. To avoid this potential problem, several modes all involving feedback are recommended by the NBS. [11] One of these is "cipher block chaining" (CBC). In this mode, each 64-bit block of plaintext is exclusive-ored with the DES ciphertext output for the *last* 64-bit data block before being encrypted. (Since there is no preceding block of ciphertext the first block of plaintext is instead exclusive-ored with a prearranged "initialization vector"). The ciphertext at a given point in the message therefore depends on *all* of the data preceding it, not just the current block. Any change therefore "propagates" throughout the rest of the message.

4. Authentication With DES

This error-propagation property is the basis of the authentication scheme, which can now be described.

Encrypt each packet with DES in the cipher block chaining mode. Then send the original unencrypted packet along with the last cipher word (64 bits) of the encrypted version. The receiver also encrypts the packet and compares its final cipher word with the received version; if they match, the packet is accepted. Changing even one bit in the message results in a completely unpredictable change in the cipher checksum with only 1 chance in 2^{64} of escaping detection by the receiver.

This technique amounts to adding a "cipher checksum" to the packet, an apt description since it functions much like an ordinary checksum or CRC. The only difference is that the "checksum algorithm" must protect against corruption or spoofing by malevolent humans as well as by nature, and therefore must be more sophisticated.

For example, it would not be sufficient to generate the cipher checksum by computing a normal checksum over the packet and then encrypting it before transmission. A spoofer could carefully construct a packet to have the same checksum as a valid packet he had seen earlier on the channel, and then append the same cipher checksum. The size of the cipher checksum is also very important. Ordinary checksums, designed to protect only against random natural corruption, are often only 16 bits wide. It is not that impractical to try all 65,536 possible checksums until the correct one is found by chance. Since the cipher checksum produced by DES is 64 bits wide, trying all possible values is out of the question. Naturally, such a wide field also provides superior protection against corruption by natural causes.

5. Playback Attacks

Hal's paper discusses in detail the problem of the "playback attack". Even though the bad guy might not be able generate his own message or corrupt a real one without upsetting the authentication mechanism, he could record and play back a valid message at a later time in an attempt to repeat the same operation.

However, using a transport (level 4) protocol designed for a datagram network neatly solves this problem. Such protocols are already designed to detect and reject duplicate packets arriving minutes or even hours after the original. This protection is necessary because a datagram network will occasionally deliver a long-delayed duplicate of a packet, usually when its routing algorithm is trying to recover from a failing or congested link. For example, TCP [12] uses 32-bit sequence numbers, so over 4 gigabytes must be sent on a given connection before the sequence numbers wrap around. In most implementations, subsequent connections generally cycle through all possible port numbers on the originating end, and this adds another 16 bits to the "sequence space". Even when the exact same pair of port numbers is reused, a clock has incremented the initial sequence number for the connection fast enough that it is very unlikely to reuse the sequence numbers from the last incarnation of the connection. In short, a properly implemented TCP can go a *very* long time before it reuses the exact same combination of source address, destination address, source port, destination port, send sequence number and receive sequence number, long enough for the DES key to have been conveniently changed in the meantime!

Connectionless, transaction-oriented transport protocols (such as might be used for the remote control of a packet switch or repeater) can protect against duplicates in several ways: with sequence numbers as in a connection oriented protocol, with timestamping (rejecting packets older than some limit), or by designing the set of commands in an "idempotent" fashion, which means that receiving a command more than once causes no further change in the state of the system being controlled.

6. Implementation Issues

How could a cipher checksum be added to TCP/IP, and how much of the packet should it cover? If the authentication is to be end-to-end, it can't include the IP header, [13] since at least the time-to-live (TTL) field is modified by each IP packet switch. The data covered by the cipher checksum must be delivered unchanged for the cipher checksum to be valid; therefore it should cover only the data following the IP header. While this would seem to open up devious opportunities based on modifying the IP header, this really isn't a problem. TCP and UDP [14] incorporate "pseudo-headers" into their checksum algorithms that include the IP source and destination addresses, protocol type and data length. Changing any of these fields in the IP header would result in a checksum error when the packet reaches the destination, and of course any attempt to modify the checksum field in the TCP or UDP header would be detected by the cipher checksum.

Another question is, where should the cipher checksum go? It would be nice to find a place to put it that wouldn't make the resulting datagram incompatible with versions of TCP/IP that didn't understand it. This suggests placing it in one of the option fields, either in IP or TCP (assuming TCP is used). If it is put in the TCP options field, one runs afoul of the specification that says options should be present only in SYN segments (connection requests). New TCP options are also discouraged. This leaves the IP options field, of which there are relatively many, some of which are not understood by every implementation. To allow for this, option codes always include the length of the option, so that an unknown option type can be skipped over. Putting the cipher checksum into an IP option also allows it to be used with transport protocols other than TCP (e.g., UDP).

Other minor issues, such as padding and the choice of initialization vector, are easily resolved. Since the DES CBC mode operates on 8-byte blocks of data, data fields not a multiple of 8 bytes long should be padded out with zeros before encryption. The initialization vector (IV) required by the CBC algorithm could serve as an additional key element; it would be then necessary to know both the 56-bit DES key and the 64-bit IV in order to generate and check authenticators. For the sake of simplicity, however, the IV should probably be standardized (e.g., all zeroes) and only the DES key used for security.

7. Summary

Authentication is far easier to implement in a datagram network than in one based on virtual circuits, resulting in a much simpler and more elegant design. Since there is only one type of packet at the datagram level, there is only one type of authentication operation. There is no need for a "session key" or "challenge" at the beginning of a connection should one exist at a higher protocol layer. Each datagram is individually authenticated to protect it against spoofing or corruption, and eliminating the possibility of a bad guy taking over a connection (assuming one exists) after it has been established. The measures already in place to protect against the accidental packet duplication possible in a datagram network automatically guard against playback attacks.

A public-domain implementation of DES in C is available from the author.

8. References

1. Feinstein, Hal, WB3KDU, "Authentication of the Packet Radio Switch Control Link", *Proceedings of the ARRL Amateur Radio 5th Computer Networking Conference*, p 5.12.
2. Federal Communication Commission rules, Section 97.117 (Codes and Ciphers Prohibited).
3. Ibid, section 97.42 1 (a) (Telecommand Operation).
4. Federal Information Processing Standards Publication 46, "Data Encryption Standard", January 15, 1977.
5 American National Standards Institute, "American National Standard Data Encryption Algorithm", ANSI X3.92-1981.
6. Davio, et al, "Analytical Characteristics of the DES", *Crypto '83*, Santa Barbara.
7 Sugarman, "On Foiling Computer Crime", *ZEEE Spectrum,* July 1979.
8. Davies, "Some Regular Properties of the 'Data Encryption Standard' Algorithm", National. Physical Laboratory, Teddington, Middlesex, UK.
9. Lexar Corporation, "An Evaluation of the NBS Data Encryption Standard".
10. Branstad et al, "Report of the Workshop on Cryptography in Support of Computer Security", National Bureau of Standards report NBSIR 77-1291.
11. Federal Information Processing Standards Publication, "Announcing the Standard for DES Modes of Operation".
12. Postel, ed, "Transmission Control Protocol Specification", ARPA RFC 793.
13. Postel, ed, "Internet Protocol Specification", ARPA RFC 791.
14. Postel, ed, "User Datagram Protocol", ARPA RFC 768.

A High Performance, Collision-Free Packet Radio Network

Phil Karn, KA9Q

ABSTRACT

For the past several years, those discussing "level 3 networking" have made much of the performance gains to be had through hop-by-hop acknowledgements. In this paper I will show that, while sometimes helpful, hop-by-hop ACKing is not the panacea it is generally perceived to be. Only fundamental changes in the way we allocate and use frequencies will really fix the problem.

1. Introduction

At present, our networks can best be described as "anarchistic." Single frequency digipeaters abound, and everyone knows just how likely you are to get a packet across five digipeater hops on a heavily loaded frequency [2]. Given this situation, software that provides hop-by-hop acknowledgements (e.g., NET/ROM [4]) is clearly a major win. Actively retransmitting ACKs, as in the ACK-ACK protocol [3] would yield an additional improvement.

Yet NET/ROM and ACK-ACK both fail to attack the fundamental problem: *carrier sense multiple access (CSMA) simply doesn't work very well on an open-access simplex radio channel.* Two things contribute to this. The first is the well-known *hidden terminal problem:* not sensing carrier on the channel does NOT guarantee that you won't interfere with someone if you transmit.

The second problem is less well known. Because it is the converse of the hidden terminal problem I will call it the *exposed terminal problem!* A station in a good location (e.g., a mountaintop) may hear local traffic from within a distant area. Not knowing that it would not interfere with that traffic by transmitting, it defers unnecessarily and wastes an opportunity to reuse the frequency locally.

In short, the carrier detect line from the modem is often useless. There is no guarantee that you won't interfere with someone if you transmit when you don't hear a carrier, and conversely there is no guarantee that you *would* interfere with another transmission even if you transmit when you *do* hear a carrier.

It is well known (and proven in practice!) that CSMA breaks down in the presence of hidden terminals, degrading rapidly to the performance of pure Aloha (where stations transmit at will, without first monitoring the channel). With the standard Aloha assumptions (many terminals each generating a tiny fraction of the total channel load) the maximum attainable channel throughput is only 18%. This occurs at an offered load of 50%, i.e., each packet has to be transmitted about 2.7 times on the average for it to be received once. Although hop-by-hop acknowledgements keep these figures from getting exponentially worse across a multi-hop path, they do *not* fix the fundamental problem: *CHANNEL COLLISIONS*[1]

This is a very important point. *Using link level ACKs to improve performance is, at best, a band-aid solution.* Because they represent overhead, sometimes they are actually counterproductive. The real challenge, therefore, is to make collisions impossible in normal operation. I will now discuss two of the traditional methods for collision avoidance when hidden terminals are present.

[1] George Flammer, WB6RAL, calls this the white lightning effect. [I]

2. Token Passing

One way to avoid collisions is to require each station to wait for explicit, one-at-a-time permission to transmit. When a station has sent its traffic, it passes this authority on to the next station. Since the message that grants permission to transmit is known as a *token,* this scheme is known as *token passing.*

Token passing works well in small networks with reliable nodes and links, but it doesn't scale well. Complex recovery algorithms must be worked out to recover from lost tokens caused either by failing nodes or transmission errors. In a packet radio network with many hidden terminals, the route that the token will take must be mapped out in advance; it cannot be passed between stations that cannot communicate. This complicates the addition of new stations to the network. In addition, much time is wasted passing the token when there are many stations in the network but only a few are actually sending traffic. Nevertheless, token passing is a completely unexplored technique in amateur radio, one that deserves serious consideration for special circumstances.

3. Busy Tone Multiple Access (BTMA)

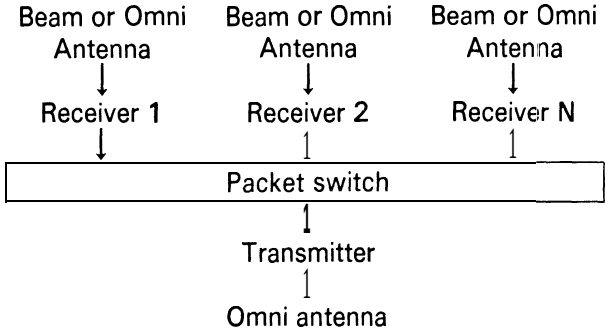
Another effective technique for eliminating collisions when hidden terminals are present is for each station to transmit a signal on a separate frequency whenever it is actively receiving a packet. If another node hears this *busy tone,* it avoids transmitting knowing that it would interfere with the reception in progress. It is not necessary for a node to couple its busy tone directly to the receiver carrier detect indication; it may drop the busy tone once it determines by examining the packet destination address that the packet is for another station. This allows *frequency reuse* (successful simultaneous use of the same frequency by two pairs of stations far enough apart not to interfere with each other).

In theory, BTMA can be an effective solution to the hidden terminal problem. However, extra radio hardware is **required** since the busy tone transmitter must operate without interfering with data reception. In practice this means using separate frequency bands, and it may be difficult to get the range of the busy tone transmitter to match that of the data transmitter -- a fundamental assumption in BTMA. It is also difficult to get BTMA to solve the exposed terminal problem. Hearing a busy tone doesn't *always* mean that you'd interfere with a receiver if its desired signal is much stronger than yours, depending on the capture ratio of the modulation method in use. Setting the busy tone's amplitude in inverse relationship to the level of the signal being received, plus lots of tricky threshold adjustments in the busy tone receivers, might make this work.

4. Contention-Free Channels

The discussion so far has centered on reducing or eliminating collisions when a single frequency must be shared by more than one transmitter. Contention channels are likely to be with us for some time where random end-users are involved. However, the emerging network of dedicated, "backbone" sites need not follow the same anarchistic model. The rest of this paper discusses a more disciplined approach that appears extremely attractive for such stations.

One sure way to eliminate collisions is to eliminate all but one transmitter on each frequency. All other transmitters on the same frequency must be placed far enough apart so that their coverage areas do not overlap. Each station uses a separate, dedicated receiver to hear each of its neighbors; it does not listen on its own transmit frequency. A network node might look like this:



Many things now become easier or perhaps even possible for the first time. As it is no longer necessary to "get off the frequency" quickly when a station has sent its traffic, fast transmit-receive switching is no longer required. Transmitters and power amplifiers with relays or slow-lockup synthesizers need not be modified; they could operate either continuously, or with tail timers like those in conventional voice repeaters. Similarly, coherent receiver demodulators (which work well with very low signal levels but require relatively long acquisition times) need not penalize network performance. The link receivers may be cheap pocket scanners since they need not transmit. If adjacent nodes transmit on different bands, the expense of repeater-style duplexers can be avoided, although filter cavities ("trashcans") may still be needed (especially at hilltop sites) to reject strong out-of-band signals.

Since the design of this network makes collisions impossible, with proper modem design and adequate RF link margins the raw packet loss rate should be very low. The occasional end-to-end retransmission of a dropped packet will be more than offset by the savings in overhead gained by avoiding link level acknowledgements. High channel speeds are much easier to handle since the packet switches are much simpler, and real time applications such as packet voice become practical. Since the nodes are inherently full duplex, sliding-window transport protocols (with data packets and acknowledgements flowing simultaneously in both directions) finally make sense, as data/ack collisions are avoided.

5. Broadcasting

In addition, some very powerful broadcast techniques become possible. Much of the traffic now handled by bulletin boards consists of undirected messages read by a wide audience. At present, our virtual circuit protocols require that a separate copy be sent to and acknowledged by every interested reader. This wastes one of the most useful and unique properties of radio: the ability of more than one receiver to hear a single transmitter. Efficient but reliable broadcasting on a very unreliable channel (e.g., an existing digipeater network) is almost impossible. However, the situation changes completely if the raw packet loss rate can be lowered to a reasonable level.

Consider the operation of an ordinary voice bulletin net, one organized to disseminate information of general interest to many stations. (A good example is the Tuesday night AMSAT net on 75 meters). After the control station finishes reading,% he invites requests for repeats. If conditions are good, only a few stations will respond, and the requested message fragments are retransmitted. As with the original transmission, all receiving stations are free to make use of the retransmitted information; this often preempts a second station's request for a fill. If conditions are bad, the control station may first read the entire bulletin several times (a simple form of forward error correction) to cut down the number of fill requests.

6. Flood Routing

Given a reasonably reliable channel (i.e., one with only a single transmitter) this scheme should be easy to automate. Wide-area bulletin coverage could be achieved with a flood routing scheme similar to the USENET bulletin board network. In flooding, a node originating a message transmits it to all of its neighbors. Each message contains a unique network-wide identifier (e.g., the node address concatenated with a serial number). Each receiving node maintains a list of messages it has already seen and ignores duplicates. A non-duplicate message is entered into the list and retransmitted to its neighbors until it has spread to every reachable node in the network.

Flooding is extremely robust, as it tries every possible route to each node in parallel. USENET has proven this in practice, despite an amazingly anarchistic network management style. It is the preferred way to reach large numbers of people, since a given message crosses each link in the network exactly once. Because of its reliability, flooding is a useful fallback for high priority point-to-point traffic when ordinary routing schemes have failed. (One often finds person-to-person messages posted on USENET because direct mail routing hasn't worked. Clearly this is to be discouraged except as a last resort because of the unnecessary load this generates.)

7. Summary

The use of contention-based channel access algorithms is perhaps unavoidable where end users are involved. However, such free-for-alls are inappropriate on backbone links in light of the severe performance problems involved. The evolving backbone networks should take a more enlightened approach. Instead of just attempting to patch things up at a higher layer by adding hop-by-hop acknowledgements, they should be carefully planned to avoid collisions altogether. Not only can the extra overhead of hop-by-hop acknowledgements be avoided, but qualitatively new and vastly more efficient bulletin dissemination techniques fall out almost for free. Considering the vastly improved performance and functionality that would result, the extra costs of doing so are minimal.

8. References

1. Flammer, G., "Survival Training for Mountaintop Digipeaters," *73 Magazine*, August 1986 p. 68.
2. Clark, T., "The Trouble with Digipeaters," *Gateway*.
3. Karn, P. and Lloyd, B.: "Link Level Protocols Revisited," *ARRL Amateur Radio Fifth Computer Networking Conference*, pp. 5.25-5.37, Orlando, 9 March 1986.
4. Raikes, R., and Busch, M., "NET/ROM Version 1 Documentation," Software 2000 Inc, May, 1987.

The KA9Q Internet (TCP/IP) Package: A Progress Report

Phil Karn, KA9Q

ABSTRACT

For over two years, the author has led the development of C-language software implementing the ARPA Internet protocol suite (commonly called "TCP/IP"). Intended for amateur radio use, the software was originally written on and for the IBM PC and its clones running MS-DOS. However, the use of a de-facto industry standard protocol set has resulted in considerable interest in and contributions to the effort by non-amateurs as well. The software has been "ported" to several different computers and is enjoying increasing use in both conventional Local Area Network (LAN) environments as well as amateur packet radio.

This paper describes the considerable progress this effort has made, and reflects on the choice of TCP/IP now that significant on-air experience has been gained.

1. Introduction

At the Fourth ARRL Amateur Radio Computer Networking Conference in March 1985, I proposed that the ARPA Internet Protocols be used in amateur packet radio. [2] [3] Since then, TCP/IP has become a reality on amateur radio. Many stations now run TCP/IP almost exclusively, and it is increasing steadily in popularity.

2. What "TCP/IP" is - and isn't: A Review

Unfortunately, the subject of higher level networking protocols in amateur radio (including, but not limited to, the famous virtual circuit vs datagram debate) is still highly controversial in certain circles. Because many misconceptions about TCP/IP persist, I will review what it is and is not.

TCP and IP are only two elements (albeit very important ones) of a larger, modular set of protocols known as the ARPA Internet Protocol Suite. It must be stressed that only "higher level" protocols are specified. In ISO jargon, the ARPA Protocol Suite begins with the upper half of the network layer (level 3B, also called the internet sublayer) and goes all the way up to level 7, the application; levels 1, 2 and 3A are deliberately left unspecified. This is in keeping with ARPA's original purpose: constructing a uniform *internetwork* out of an existing collection of dissimilar links and even entire networks that would otherwise be incompatible with each other. There is no "ARPA Standard Link Level Protocol" because there is no need to require only one; they can all coexist yet still interoperate.

AX.25 Level 2, X.25 and the network layers of NET/ROM, Texnet, and even COSI fit into the Internet model very nicely. TCP/IP does *not* compete with these developments (except for the level 4 or transport level components in some of them), but rather *complements* them all. By filling a very real need, the ARPA suite is today *the* de-facto industry standard for computer networking when a wide variety of computers and underlying networking technologies must be interconnected.

3. What "Higher Level Networking" Really Means

In the ISO model, everything above layer 3 is "end-to-end". That is, layers 4 through 7 exist *only* in the end users' machines *(hosts* in ARPA terminology, *end systems* in ISO) not in the intermediate packet switches *(gateways).* Unfortunately, some have used terminology at variance with this definition, causing considerable confusion as to the meaning of "higher level networking". For example, the addition of hop-by-hop acknowledgements to a network is *not* "level 3 networking", it is a level 2 function. Further, by strict interpretation of the ISO model, we already have a de-facto datagram-oriented

"network layer" in the address field that is part of "AX.25 Level 2". We already have a de-facto "transport" protocol running in the TNCs that maintains connections on an end-to-end basis. Installing "real" level 3 and 4 protocols means pushing AX.25 back down to the link layer it was designed for.

These distinctions are not just semantic quibbling. They affects one's entire view of how the pieces should fit together in the network and how it should appear to the users.

4. Computer Networking vs Terminal Networking

It should now be apparent that the purpose of higher level protocols is to connect *computers,* not just terminals, and to use the capabilities of those computers effectively. (Running a "terminal program" on a PC is *not* using it effectively.) True networking is much more than just providing "virtual wires" between dumb terminals[1] or even a dumb terminal and a bulletin board system. While packet's use of faster modems, addressing, error detection and retransmission does provide channel sharing and error-free communications, without higher level protocols running on the user's computers the result is *qualitatively* little different than ordinary radio teletype (RTTY).[2] In contrast, *true* networking involves one or more high level (application, presentation and transport) protocols plus a multi-tasking operating system running in the end-user's machine, performing end-to-end functions automatically on behalf of human users. For example, a single system might respond automatically to remote requests for file, accept remotely sent files and electronic mail for storage, and initiate similar operations on other computers in response to local commands -- all simultaneously, with minimal operator intervention. In the following sections I will review the major ARPA protocols and describe the implementation of those in the KA9Q Internet software package, starting with the top layers and working down.

5. Internet Software Components

A major goal of the KA9Q Internet package was that multiple network activities should go on simultaneously. This greatly increases the usefulness of the system, and alleviates many other potentially troublesome problems. For example, "stuck connections" caused by the other station abruptly disappearing are only a minor annoyance, as the only resources wasted are a few bytes of RAM. The system can still be used by others; keepalive timers aren't necessary.

Since MS-DOS is not a multitasking system, all of the protocols were combined in a single MS-DOS program called **net.exe,** and a very simple form of multitasking is performed internally by a "commutator loop" mechanism.[3] The main loop of the program sequentially polls various routines to see if they need service. For example, the keyboard is checked for input, then the serial input buffer is checked for characters, then the Ethernet receiver is checked for packets, then the timer is checked to see if a tick occurred, and so on.

When events occur, calls are made to the appropriate routines; incoming data triggers the appropriate link level protocol modules, which in turn call the higher level modules, and eventually the applications are called. This is known as an *upcall* or *pseudo-interrupt* mechanism, and has been used for other small networking packages such as MIT's PC/IP. It is important to note that there is no conventional sleep/wakeup mechanism; each application must provide functions to be called asynchronously by the system. These functions cannot block or hog the processor; they must respond to the event and return. The applications must therefore be structured as state machines driven by upcalls. While this is a somewhat unusual environment for a programmer, it isn't too hard to get used to it. In fact, it encourages the programmer to think about what should happen for every possible combination of state and event; it's easy to get sloppy about this in a conventional environment by assuming that only the desired event can occur in a certain state.

---

[1] The term dumb terminal was originally a trademark of Lear Siegler Corporation for their ADM-3 CRT terminal. The term quickly became generic, referring to any keyboard/display (or printer) combination lacking programmability and local file storage. Only recently has it really become pejorative, since many complete personal computers now cost significantly less than dumb terminals.

[2] Some call conventional packet operation "RTTY packet".

[3] This is somewhat similar to the multitasking form of FORTH known as IPS, or Interpreter for Process Structures. IPS was developed by DJ4ZC for use in the AMSAT Phase 3 satellites.

I do not mean to imply that the commutator loop approach is superior. I originally chose it as a simple expedient, and since then I've been surprised by how much I've been able to do with a very small amount of memory. The entire executable program net.exe, containing all of the protocols about to be described, is only about 60K bytes. In comparison, the popular PC terminal program "Procomm" is almost three times as large![4] One major drawback of the upcall structure, however, is the lack of application portability. Future developments may include a true multitasking kernel so that a more conventional programming environment may be supplied as well.

The user interface is simple, but functional. Commands to invoke each of the applications are provided, along with others primarily useful for monitoring and statistics gathering. Up to ten client "sessions" may exist at any one time, and the user may switch between them at will. There is no limit on the number of server sessions that may exist at one time other than the memory available on the machine for buffering and housekeeping.

5.1. Telnet, FTP and SMTP Applications

While many application protocols have been built for packet networks, three are most useful: remote login, file transfer and mail transfer. This is reflected in the "big three" ARPA Internet application protocols: Telnet, FTP and SMTP respectively. Each application protocol is further broken down into *client* and server halves. Clients act on behalf of local users by initiating communication with remote servers that passively await their requests.

Clients and servers for all three protocols are presently in the software, although the telnet server does nothing more than route an incoming connection to the console for keyboard-to-keyboard chatting. (MS-DOS isn't a timesharing system, so it wasn't possible to provide conventional telnet service). FTP supports both ASCII (default) and binary file transfers, with passwords protecting against unauthorized file access. SMTP is presently functional but it does not have the features of mailers found on larger systems such as mailing lists and mail-level forwarding. Two miscellaneous application servers are also provided: echo and discard. They are intended mainly for testing.

5.2. TCP and UDP Transport Protocols

The applications just described all use TCP, the Transmission Control Protocol. TCP is a transport/session layer (level 4 and 5) protocol that provides virtual circuit services on an end-to-end basis. The use of TCP is not mandatory, however. Some important applications prefer not to use virtual circuits, so an alternative is supplied: UDP, the User Datagram Protocol. UDP is important for routing algorithms, information broadcasting and transaction-oriented applications such as the Network File System (NFS).[5]

The TCP was the first module written for the package and is now quite mature. Three upcalls are provided to the application layer: receive, send and protocol state change. The receive upcall indicates that data has arrived which may be read from the receive queue by the application. The send upcall indicates that outgoing data has been acknowledged, freeing up buffer space that may now be used for additional transmissions. The protocol state change upcall indicates when connections are opened and closed, and applications use these to drive their own state machines and to determine end of file. Much effort has gone into tuning the TCP retransmission algorithms for efficient operation across amateur packet radio, including a novel approach to measuring round trip times accurately during periods of high packet loss. [6]

The UDP module is much simpler than TCP. Only a receive data upcall is provided.

---

[4] I do admit that net.exe lacks certain essential features, e.g., exploding windows and sound effects.

[5] NFS is not implemented in the package (yet) so it will not be described here. However, its popularity is mushrooming around the Internet. On many Ethernet local area networks (LANs) NFS/UDP traffic now exceeds that using TCP.

5.3. Internet Protocol (IP)

The core protocol of the ARPA Internet is called, naturally enough, the Internet Protocol (IP). IP sits immediately above the existing lower level networks (*subnets* in ARPA terms). This is the only mandatory protocol in the entire suite. The higher level protocol in use (TCP, etc) need be agreed upon only by the hosts involved, but you can't be "on the Internet" unless you run IP.

Since IP is "spoken" by hosts and "interpreted" by the gateways (packet switches), I found it convenient to split my IP into two halves. The upper half contains the parts of IP relevant to a host (outgoing packet generation, incoming packet demultiplexing, fragment reassembly). The lower half contains the IP packet switching components (routing and fragmentation). Every host is also a gateway, i.e., it will route and switch traffic that is just passing through in addition to sourcing and sinking its own traffic. If the code is to be run on a dedicated gateway, the host functions can be removed to save space.

The packet routing portion of my IP includes a generalized form *of subnetting*, the ability to structure the address space into a tree-shaped hierarchy. [3] Each entry in the routing table includes a width field saying how many bits in its address field are significant. When packets are routed, the algorithm finds the routing table entry providing the "best match" to the leading bits of the destination address. This allows large blocks of addresses that share a common next hop, e.g., all west coast addresses in an east coast switch, to share a single routing table entry. This reduces the average size of a routing table enormously while still allowing arbitrary routes to be set up. Even this relatively complex technique, however, only takes about 6 milliseconds to route a packet on the IBM PC, thereby refuting the argument that datagram routing "costs" too much.

No automatic routing algorithm is provided as yet; the routes must be set up manually. Automatic routing is clearly a desirable long-term goal, but it is a difficult and challenging problem in any network so I have deferred it.

5.4. Subnet Protocols

Link/subnet drivers for AX.25 Level 2, Ethernet and asynchronous point-to-point links are presently provided. In the spirit of the Internet, others can be added easily as they become available (e.g., NET/ROM, Texnet and COSI).

The AX.25 Level 2 driver is at present very simple; each IP datagram is encapsulated in a single AX.25 UI (connectionless) frame. The KISS TNC [1] was developed to allow these "raw" packets to be generated. This was an expedient for initial operation; it is *not* the only way that IP can be run on top of AX.25. It is entirely possible to use the full-blown connection-oriented AX.25 protocol under IP if necessary to improve hop-by-hop reliability; this will have to be done to use the internals of NET/ROM to move IP traffic, for example. On the other hand, collision-free backbone channels would do well to avoid the extra complexity and overhead of link level acknowledgements.

It is incorrect to say that IP cannot be run efficiently on noisy poor channels because of its relatively large header size. While not done at present, the subnet or link may perform intranet fragmentation. That is, it may chop up a single datagram into multiple smaller packets and reassemble them transparently at the other end of the link before passing them up to IP. This is distinct from the internet level fragmentation done by IP, and is preferable for performance reasons when the subnet has an unusually small packet size limit. As for the "inordinate" overhead associated with a 40-byte TCP/IP header, consider that even at 1200 baud, 40 bytes takes only a quarter of a second to send. Many stations spend more than this just keying up the transmitter. As faster modems (e.g., the new WA4DSY 56 kbps design) become widespread, header overhead will become even more of a non-issue.

5.5. The Address Resolution Protocol (ARP)

The Internet Protocol provides its own addressing independent of that used in the subnetworks. It is therefore necessary to map IP addresses into subnet addresses for each hop. Sometimes this can be done by making the subnet address part of the IP address, but frequently this isn't possible because the subnet address is too big. This is the case with both Ethernet and AX.25, so the Address Resolution Protocol was implemented. [7]

The ARP module in the package serves both subnet protocols. Since ARP requires a broadcast facility in the subnet, I chose "QST-0" as the AX.25 broadcast address. Manual commands are provided to manipulate the ARP translation table, either to override the automatic mechanism or to specify multi-hop digipeater paths. The latter must be done manually since the AX.25 subnet can no longer provide broadcasting when digipeaters are involved.

6. Availability

The entire software package, including executable programs, complete source code and documentation, is available to interested amateurs for the cost of copying only. It may be freely used and copied for noncommercial purposes only. Brian Lloyd, WB6RQN, handles the distribution on MS-DOS format floppy disks; send him $5 to cover costs and he will provide disks, mailers and postage. Persons outside the US should add enough to cover higher postage costs.

7. Credits

Many people have contributed in various ways to this effort, so what follows is necessarily only a partial list. Bdale Garbee N3EUA wrote the mail command *bm,* and coordinates the integration of software releases. Mike Chepponis K3MC wrote the first KISS software for the TNC-2 and has been instrumental in encouraging others to support other TNCs (see [1] for a complete list). Jon Bloom KE3Z contributed the driver for the HAPN HDLC adapter card for the PC. Brian Lloyd WB6RQN has widely promoted the amateur use of TCP/IP, writing introductory magazine articles geared to the novice user. When the code first became available, Brian organized a local group of users in the Washington DC area that has since grown to several dozen; their feedback, as well as that of other groups around the world, has proven very useful in improving the package. Brian also spends a considerable amount of time distributing the package on floppy disk by mail.

While I wrote the bulk of net.exe myself, others supported my efforts by patiently answering my many questions about the details of the protocols. Thanks go to Dave Mills W3HCF of the University of Delaware and Jon Postel of the University of California Information Sciences Institute.

8. References

1. Chepponis, M., and Karn, P., "The KISS TNC: A simple Host-to-TNC communications protocol," this conference.
2. Karn, P., "TCP/IP: A Proposal for Amateur Packet Radio Levels 3 and 4," Fourth ARRL Amateur Radio Computer Networking Conference, San Francisco, March 1985, p. 4.62.
3. Karn, P., "Addressing and Routing Issues in Amateur Packet Radio," Fourth ARRL Amateur Radio Computer Networking Conference, San Francisco, March 1985, p. 4.69.
4. "For own in-house network, COS selects TCP/IP," Data Communications magazine, June 1987.
5. Bloom, J., "NET.EXE: Beyond the TNC," Gateway Vol. 3 No. 12, Feb 6, 1987.
6. Karn, P., and Partridge, C., "Improving Round-Trip Time Estimates in Reliable Transport Protocols," SIGCOMM Workshop proceedings, Stowe, Vt., August 1987.
7. Plummer, "Address Resolution Protocol," ARPA RFC 826.

APPROACH FOR DIGITAL TRANSMISSION OF PICTURES

by Thomas Kiesel bach , DL2MDE
(translated by Don Moe, DJ0HC/KE6MN)

"A picture is worth a 1000 words" is a saying confirmed daily. It should be remembered that transmission of picture signals began in the early days of radio. The individual techniques were adapted to the specific needs at the time, thereby resulting in diverse system5 each being optimized in accordance with the requirements and the method of transmission used.

Starting with what we perceive with our eyes, compromises are necessary in all picture transmission systems in order to:

1. limit the information to be transmitted,
2. direct the user's attention to the most essential information.

Television is an example of the first point. In the case of terrestrial transmission, there is just not enough bandwidth on the available frequencies to permit higher resolution5

FAX transmission is an example for the second point, where limiting the data to black and white provides a better understanding of the information.

There is another aspect, typical in all transmission techniques, including picture transmission. Initially the transmission was implemented using analog signals. Only recently have digital methods become more widely used, since a computer is required to support this technique fully. A major advantage of digital transmission is securing the information aqainst interference. This presumes that more information must be transmitted than is the case in analog transmissions.

Regarding the informational content of the pictures, there are two areas which must be considered as separate applications:

1. a still picture which can be studied at length, such as a photo,
2. a motion picture, such as film or television, which provides temporal continuity,

Errors in the pictures or their transmission can be tolerated to a certain extent in the case of motion pictures, since redundancy is assured by the quantity of the pictures, and because the observer cannot study each individual frame down to the finest detail in such a short time. In the case of a still picture, each individual picture element (pixel) is important however. It is remarkable that so much more attention has been paid in recent years to the transmission of motion picture5 than of still pictures, considering that a single still picture frequently conveys more information than does a motion picture,

In the case of analog picture transmission, the picture to be sent is sampled line by line. At the start of the transmission a synchronization signal indicates the start of a frame. Thereafter each line proceeds with a synchronizing pulse and the content of the line in analog form. The conclusion of the frame is marked by either ending the transmission or by sending the synchronizatian signal for a subsequent frame. This system is used in all techniques, though there are obviously variation5 in the technical implementations in each system. A digital system functions according to this same principle, except that the synchronization pulses and characters can contain additional information, and the picture content is transmitted in digital form. In the following article, the emphasis is on the transmission of still pictures, and a suitable application is outlined. The inclusion of digital picture transmission into data nets for packet radio will likewise be described.

Preparation of the Picture

The picture is captured using an electronic camera. For transmission of motion pictures via television, various systems have been developed, whereas for still pictures there are only a few dedicated to very special application5 Therefore such a camera is generally employed in this case. The difference lies in the time needed by the camera to capture the picture. The standard is 1/25 of a second, as defined by the television norms. Motion distortion is uncritical in television, though for still pictures it is an error that is very difficult to correct. Errors of this type arise not only through motion of the photographed object but also due to motion of the photographer. Fortunately this error doesn't result in poor focus from the rapid motion, as is the case in normal photo, as long as the interlaced frame technique of alternate line scanning is not used. In the case of tube cameras, the conversion to

non-interlaced operation is difficult, thouqh feasible for CCD cameras.

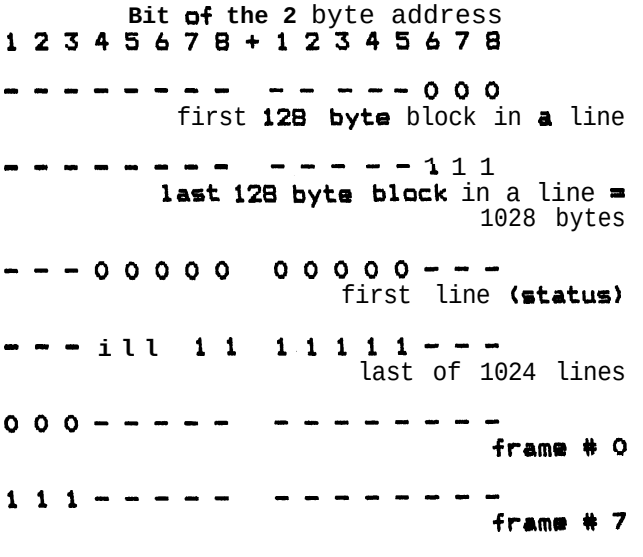
There are various output standards for the information from an electronic camera, although they will not be discussed here. The following discourse initially presumes a black and white camera supplying analog data during the output of a full frame without interlaced scanning. Techniques of data compression are not considered at this time.

The first step is the transfer of the picture information into the memory of the transmitting station. This process must occur within 40 ms. Two processes are taking place in parallel at this point=

1. forming the memory addresses depending on the picture and line synchronization, and
2. digitizing and staring the picture information.

For both processes, the primary decision to be made is the resolution of the stored picture. The resolution must equal or exceed that used in the receiver. Even when it won't initially be used, it is advantageous to work with an arrangement supporting a large number of pixels and possibly to ignore a portion of them. A matrix of 1024 x 1024 (512 x 512) is suitable for storing a high resolution picture.

The technique must also permit sampling of portions of the picture by line and identifying them in the transmission. This can be done with a two byte address word containing three different data items. The first three bits are used as the frame counter, bits 4-13 are the line counter, and the remaining bits, 14-16, form a block counter within a line. The three portions of the J-bit address word are loaded from three binary counters, in order to keep the computational time short for the creation of the address.



The frame counter in the address field can be used for several tasks: a) the consecutive numbering of the pictures being transmitted, or b) the identification of special features, ie. marking frames with the red, green or blue color segments.

If the address field is not fully used, as will be the case for most of the common installations, the corresponding address bits will then be clear, and the memory locations will be unused or vacant.

For the conversion of the analog picture signal into digital data, a fast A/D converter is required. Appropriate components are readily available on the market for this application. The LSI component TDC1025 from TRW has been implemented in a trial circuit. It is a monolithic high-speed analog-to-digital converter with parallel output, using B-bit flash architecture with 50 million samples per second. Using this device, a resolution of 512 x 512 pixels is easily attained.

The picture stored according to this method is then available in the RAM memory for transmission, or it can be superimposed on other carrier signals.

Transmission of the Picture

Depending on the resolution of the picture, each frame results in a corresponding volume of data. For a black and white picture, these amounts are as follows, in accordance with the resolution:

128 x 128 Pixel with 8 bit resolution = 16.3 KByte

256 x 256 Pixel with 8 bit resolution = 65.5 KByte

512 x 512 Pixel with 8 bit resolution = 262.1 KByte

1024 x 1024 Pixel with 8 bit resolution = 1048.6 KByte

In addition the synchronization information must be supplied for control and data security.

This quantity of information can be significantly reduced by techniques of data compression without causing significant degradation in the resolution of the picture. Testing and describing the various techniques are reserved for a subsequent investigation and should initially be employed in connection with the expanded data volume for color pictures. Through the use of various data compression techniques, color pictures can be transmitted with the same data volume as uncompressed black and white pictures. Since no other procedure has been established, the transmission of color pictures will be in frames containing the three component colors, red, green, and blue.

Mother point to consider is the transmission of the necessary overhead information. This consists of the synchronization word per block (24 bits), the control information, such as memory addressing, and an error recognition code of a 16 bit frame check sequence (FCS) or the implementation of a BCH code, that can correct at least three errors and recognize even more due to 40 additional bits. Consequently the transmission overhead lies between 5 and 10%.

When discussing error recognition or correction, the operating mode used by the picture transmitter needs to be additionally considered. In the case of a paint-to-point connection, error recognition including determination of the number of errors per block is preferable, since incorrect blocks can be requested following transmission using the ARQ method. For broadcast transmission on the other hand, error recognition does not provide any advantages, whereas beyond a certain number of errors, error correction does,

The main criterion for the transmission of individual pictures is the available bandwidth of the, transmission channel. Direct relationships exist between the bandwidth of a channel, the transmitted resolution of a picture, the modulation method, and the duration of a transmission.

Speed	Resolution 128x128	256x256	512x512
4 kB/s	35.3+	141.0	565.0
8 kB/s	17.6	70.6	282.0
16 kB/s	8.0	35.3+	141.0
32 kB/s	4.4	17.6	70.6
65 kB/s	2.2	8.8	35.3+

Transmission time in seconds

Using an experimental system, a first attempt was made by transmitting the pictures in increasing resolution and at different speeds. In this case, the picture is transmitted initially every eight pixel in every eight line at a speed of 4 kB/s. This corresponds to a low resolution (LR) picture of 128x128 pixels after decoding at the receiving station.

The next step is enhancing to a medium resolution (MR) picture by transmitting the missing pixels at a speed of 16 kB/s, for a resolution of 256x256. Following this phase of the transmission, every fourth pixel in every fourth line is present.

The third step is advancing to a high resolution (HR) picture in the same form at 64 kB/s, so that every second pixel in every second line can be displayed.

The last step, increasing the resolution of the picture to a very high resolution picture (VHR), involves considerable expense and is not considered in this design.

In accordance with the entries in the previous table, the transmission time of a picture amounts to 88.25 seconds. For the table entries marked with "+", only 75% of the times can be taken into account at the higher speeds, since a portion of the picture has already been transmitted.

Control over the transmission and access to the picture data at-e easily accomplished due to the addressing scheme.

Upon completion the transmitted picture with its line and block addresses appears as follows, where the pixel for 128x128 are marked with #, those for 256x256 with *, and those for 512x512 with +.

Phase 1 at 4 k&/s - 128*128 Pixel

'Line/Block

8 / 1	#	#	#	#
16 / 1	#	#	#	#
24 / 1	#	#	#	#

Phase 2 at 16 kB/s - 256*256 Pixel

Line/Black

8 / 2	#	*	#	*	#	*	#
12 / 1+2	*	*	*	*	*	*	*
16 / 2	#	*	#	*	#	*	#
20 / 1+2	*	*	*	*	*	*	*
24 / 2	#	*	#	*	#	*	#

Phase 3 at 64 kB/s - 512*512 Pixel

Line/Block

8 / 3+4	#	+	*	+	#	+	*	+	#	+	*	+	#
10 / 1+2+3+4	+	+	+	+	+	+	+	+	+	+	+	+	+
12 / 3+4	#	+	*	+	#	+	*	+	#	+	*	+	#
14 / 1+2+3+4	+	+	+	+	+	+	+	+	+	+	+	+	+
16 / 3+4	#	+	*	+	#	+	*	+	#	+	*	+	#
18 I 1+2+3+4	+	+	+	+	+	+	+	+	+	+	+	+	+
20 / 3+4	#	+	*	+	#	+	*	+	#	+	*	+	#
22 / 1+2+3+4	+	+	+	+	+	+	+	+	+	+	+	+	+
24 / 3+4	#	+	*	+	#	+	*	+	#	+	*	+	#

The procedure in which the picture is

transmitted with increasing transmission speed and resolution was chosen for several reasons:

1. At the lower speed and with identical channel characteristics, correspondingly better transmission conditions are achieved. If the channel is designed for the highest transmission speed, assuming an error rate of 10-3 to 10-4 at 64 kB/s, then error rates of 10-6 to 10-7 at 4 kB/s and 10-5 at 16 kB/s result. Corrective algorithms based on the appropriate evaluation of the data can be implemented at the receiving stations, taking into account the results from the slower and hence less error-prone phases. Such algorithms are especially applicable when the volume of data is reduced through data compression techniques. In any event, interference to blocks during transmission don't always result in disruption, but rather appear merely as picture areas with reduced resolution.

2. In implementing the necessary modems, simplifications can be applied since initial synchronization occurs at the slower speed and at transition to the higher speed, the clack/4 is already known, hence eliminating the need for a full synchronization phase.

3. The degree of complexity in the transmitting and receiving stations can vary. A simple receiving installation, with a modem for only 4 or 16 kB/s or lower computer capacity, can receive the pictures at reduced resolution. In the other direction, simple installations can also process and send pictures at lower resolution.

4. Techniques of data compression are deliberately ignored at this point, since premature rules always result in constraints on further development. This would not yet be desirable, especially in view of the project described next.

Reception of the Picture

A receiving station will basically consist of the following three functional components:

1. Receiver with appropriate modems,
2. Computer for picture storage,
3. Picture display via screen and/or hard copy.

The receiver must conform to the requirements of the transmission parameters. The experimental system under discussion has a channel bandwidth of 200 kHz and a DPSK demodulator for 64 kB/s. The choice of the highest transmission rate of 64 kB/s is based on the current availability of suitable components for ISDN nets.

The task of the computer is first to store the on-line data and possibly the picture display. It can additionally handle error correction and picture processing. The presently available standard PCs having a 80286 processor are adequate for the task. A memory expansion sufficient to permit handling and processing complete pictures should be provided.

The direct display of the pictures by the computer using the currently available graphic cards, i.e. EGA, is only possible with reduced picture quality. With special cards or additional display equipment, all options are open.

A specially constructed picture display driver accepts and stores the picture data from the modem or computer, and produces the necessary signals for the monitor.

Transmission in Packet Radio Nets

The design of the data structure for digital picture transmission resembles the frame layout currently used in packet radio nets. If picture transmissions are to be incorporated into the current nets, several questions need to be answered and agreed upon.

The data quantity to be transmitted is very large and would completely disrupt existing traffic in most nets, Just the transmission times disregarding the additional overhead needed by the packet radio protocol practically forbids implementation at speeds less than 9.6 kB/s. At this speed, the transfer time just for the pure picture data already amounts to four minutes. Nets of this type, running at even higher speeds, will be installed on the UHF/SHF bands in the future.

A further point is the overhead due to the packet radio protocol. The present day nets are oriented around datagramms. For picture transmission it appears more sensible to incorporate virtual connection techniques.

Another question concerns the handling of transmission errors. Different than for example in file transfers, a certain error rate can be tolerated in picture transmissions. It therefore seem5 reasonable to use error recognition such as BCH coding during validation of the received data rather than FCS control. The content of a packet would then only be requested when a permissible number of errors is exceeded or a portion of the data is missing, which can be determined by the address data.

An Interesting Project

During flight STS 61 A of the space shuttle and the first German Space Lab mission (D1), amateur radio was present as station DPOSL, for which I was responsible. Unfortunately due to the tragic accident of the space

shuttle *Challenger*, the date for a subsequent German mission has been postponed, and hopefully not canceled.

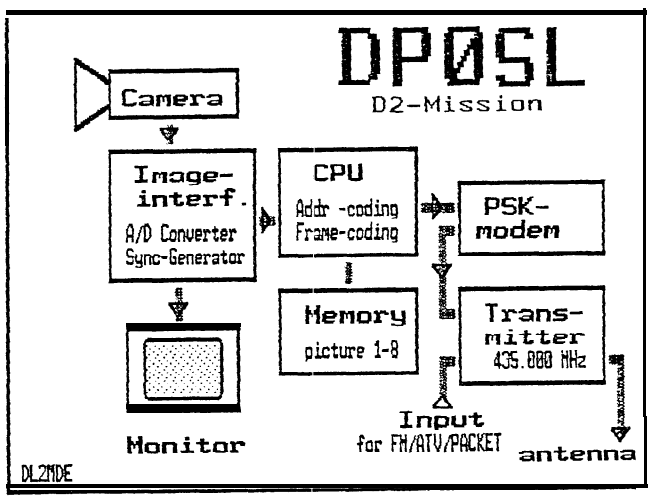
For the upcoming mission, we wish to have even more ambitious equipment aboard, and the digital picture transmission just described has been specified during the planning stage. Should this project be carried aut, positive result5 would consequently result for other applications. This is also a reason to include other interested circles early in the planning, since we do not wish to perform an experiment for only a small group. We would be quite pleased if through this project a unified standard for digital picture transmission would result, since a mission of this nature would offer the best conditions to establish one.

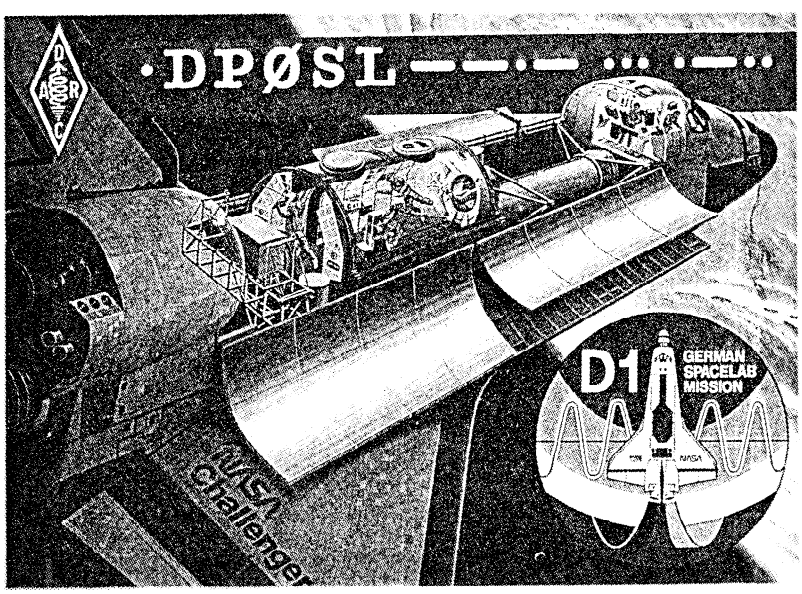
Fur DPOSL, picture transmission equipment will be constructed consisting of a camera, memory for four to eight pictures, and a 435 MHz transmitter. When the operators aboard Space Lab find the time, they will take up to eight pictures, that will then be continuously transmitted in sequence, when the 435 MHz transmitter is not required for other tasks. The transmission of one picture takes approximately 1.5 minutes, so that all pictures can be received during most passes. Thanks to the addressing scheme, missing picture5 or disrupted Portion5 of pictures can be captured and inserted during subsequent passes, unless the operator at DPOSL has taken some new pictures in the interim-

In total, four goals are under discussion in the group which previously implemented DPOSL:

1. improved voice communications,
2. ATV connections in both directions,
3. packet radio, possibly with satellite interlink,
4. picture transmission, the project just described.

I believe these goals have a good chance, since besides getting some really interesting pictures it will provide significant technical enrichment, which is the life-blood of an experimental radio service. But one other point speaks for this project= from experience we know that the operators in space have very little spare time and for this project they only need a few seconds per day,





Range of sight of' D P Ø S L in Spacelab while crossing the control center in Oberpfaffenhofen at the D1 Mission.

RUDAK - THE PACKET RADIO EXPERIMENT ON-BOARD OSCAR P3C

Hanspeter Kuhlen DK1YQ
AMSAT-DL
Program Manager AMSAT-RUDAK Project

Finkenstrasse 11
D-8011 Aschheim nr. Munich
West- Germany
Tel.: 49 89 - 90 33 90 5 (home)
49 89 - 6000 - 3542 (qrl)

Abstract

Nearly two years ago, AMSAT-DL initially published the outcome of its first RUDAK system design meeting at the 4th ARRL Conference in San Francisco [1]. At this time we wish to present the current status of the flight ready hardware and the completed and tested ground station equipment.

This paper will briefly present the objectives of the RUDAK experiment and the performance achieved by the RUDAK transponder on-board OSCAR Phase 3 C. It will also include a description of a recommended user terminal which can be utilized for all satellite operating modes via OSCAR Phase 3 C, FUJI OSCAR 12, as well as for terrestrial packet radio.

Introduction

Early in 1985 when the packet radio revolution began spreading around the world, following acceptance of the AX.25 as an international amateur radio protocol for link control, the first system design meetings took place, where the payload capabilities of the new AMSAT OSCAR satellite were discussed.

The experiment was named RUDAK which is the accronym for " Regenerativer Umsetzer für Digitale Amateurfunk Kommunikation " which is translated into English " Regenerating Transponder for Digital Amateur Communications ".

Objectives of the RUDAK experiment

These are:

* Real-time digital communications facility with ALOHA (i.e. uncoordinated) type of time division multiple access and continuous time division multiplex (i.e. permanent) on the downlink. Due to the long visibility of the spacecraft because of its elliptical orbit, a mailbox or any other store-and-forward facility was not considered necessary.

* Operational use of binary shift keying (BPSK) with differential coding for the purpose of individual communication, education and experimentation.

* General information broadcast in AX.25 (UI-frames) or AMSAT format.

* Computer controlled mode switching (autonomous operation)

*Provide in-orbit facility using a fully programmable computer with intelligent hardware interfaces in order to provide a testbed for communications experiments such as testing alternative access procedures, different bitrates and so forth.

* Keep size and complexity of ground user equipment as low as possible to motivate home-brewing of station equipment.

* Information system through a ROBOT

mode to RUDAK processor. In this mode, connects from individual stations to RUDAK will be responded with a particular message e.g. Keplerians and then disconnected. So at least a positive confirmation of a contact will be provided.

The RUDAK space segment

The RUDAK mode will be in operation on mode L only. Due to the fact that the RUDAK transponder is basicly independent from the normal passband it will be operative even when the mode L passband is switched off. The uplink frequency will be 1269.725 MHz and the downlink 435.725 MHz, however the exact frequency plan will be published after final calibration of all transponders.

A block diagram of the digital RUDAK transponder is given in figure 1.

The transmissions from the ground stations will be short packets or bursts to allow time sharing of the frequency among several users simultaneously. A specially developed burst demodulator on-board the satellite scans around the center frequency to cope with uncertainties of the uplink signals in terms of frequency accuracy, doppler shifts and so on. A range of +- 7.5kHz is scanned at 120msec/scan.

The measured performance of the flight unit is about 100 msec for the acquisition i.e. the time span from detection of an input signal to its demodulation. The demodulated signal then is the recovered data and clock for further processing in the RUDAK computer.

This unavoidable overhead time of 100msec has to be considered in setting the appropriate TNC parameter. For instance the AXDELAY in TNC 1s so, that a sufficient number of flags (#7E) are being transmitted prior to each packet thus allowing the demodulator to lock on the signal. The optimum values will be communicated in the broadcast beacons, however according to our present experience they will probably be almost the same as used for the standard FM MODEMs in the TNCs.

Due to the inherent collision problem for the uplink packets, the maximum theoretically achievable throughput is limited to 18%. In other words, even under optimum channel conditions, only 18% of the packets offered to the channel will survive. This effect is also to be observed when a mountain top packet radio digipeater is within reach of many local stations, where on the other side the local stations cannot hear each other. In this case the carrier sensing mechanism (CSMA), which normally avoids collisions by inhibiting transmissions on an engaged frequency, is useless. Therefore 2400bps has been selected for the uplink bitrate resulting in 400bps for the downlink.

As an option there is a 1200bps mode, where NRZI coding is used so as to be fully compatible with the FUJI OSCAR 12 downlink.

Mode L has been designated to be the dominating mode throughout the entire mission of Phase 3C. For the digital experiment, the RF modules will be independent of the normal passband of mode L.

RUDAK User Terminals

For the user, probably the most interesting part of the experiment is the design and installation of his/her own satellite user terminal.

As defined in the objectives, one of the main goals of the experiment is to enable reasonably skilled individual operators to test the modem modes of digital communications. This has been considered not only in the selection of moderate bitrates, but also in the design and development of an extremely versatile easily built user terminal.

During the last couple of months, all necessary modules (PCBs) for the RF and digital unit have been designed and tested. Wherever possible, we took off-the-shelf designs in order to avoid re-inventing the wheel. The PCBs are now available and have been beta-tested by several amateurs.

Now its time to bring it all together in such a way as to be easily and reliably reproduced. The design of the terminal consists of two separate cabinets, called 'RF-Unit" and "Digital Unit".

The features can be summarized as follows:

* Operates in ALL satellite modes: CW, SSB, PSK through passband and of course RUDAK modes.
* The RF-unit can also be used as general purpose power amplifier for terrestrial 23cm use, etc.
* The power amplifier provides 20W CW output power on 24cm with hybrid PA module.
* Converts 2m into 24cm; built-in attenuator to accept 1 W of driving power.
* Modulator for 2400bps
* BPSK demodulators for 400bps and 1200bps with Bi-phase and NRZI coding respectively.

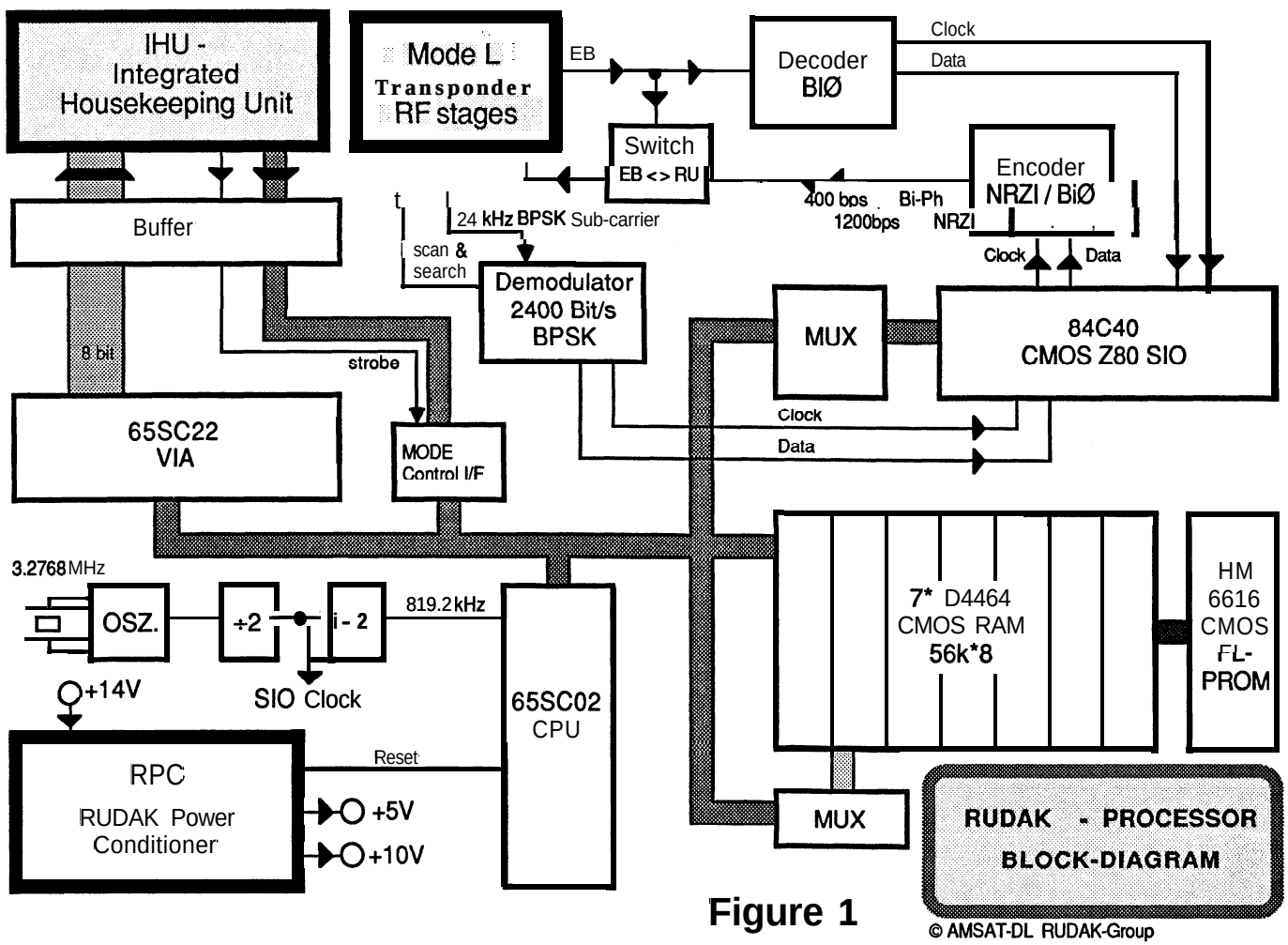

IHU -
Integrated
Housekeeping Unit
Mode L
Transponder
RF stages
EB
Decoder
BIØ
Clock
Data
Switch
EB <> RU
Encoder
NRZI / BIØ
400 bps
1200bps
Bi-Ph
NRZI
Buffer
24 kHz BPSK Sub-carrier
scan &
search
Demodulator
2400 Bit/s
BPSK
Clock
Data
MUX
84C40
CMOS Z80 SIO
8 bit
strobe
65SC22
VIA
MODE
Control I/F
Clock
Data
3.2768 MHz
OSZ.
÷2
i - 2
819.2 kHz
+14V
SIO Clock
65SC02
CPU
7* D4464
CMOS RAM
56k*8
HM
6616
CMOS
FL-
PROM
RPC
RUDAK Power
Conditioner
Reset
+5V
+10V
MUX
RUDAK - PROCESSOR
BLOCK-DIAGRAM
© AMSAT-DL RUDAK-Group

Figure 1

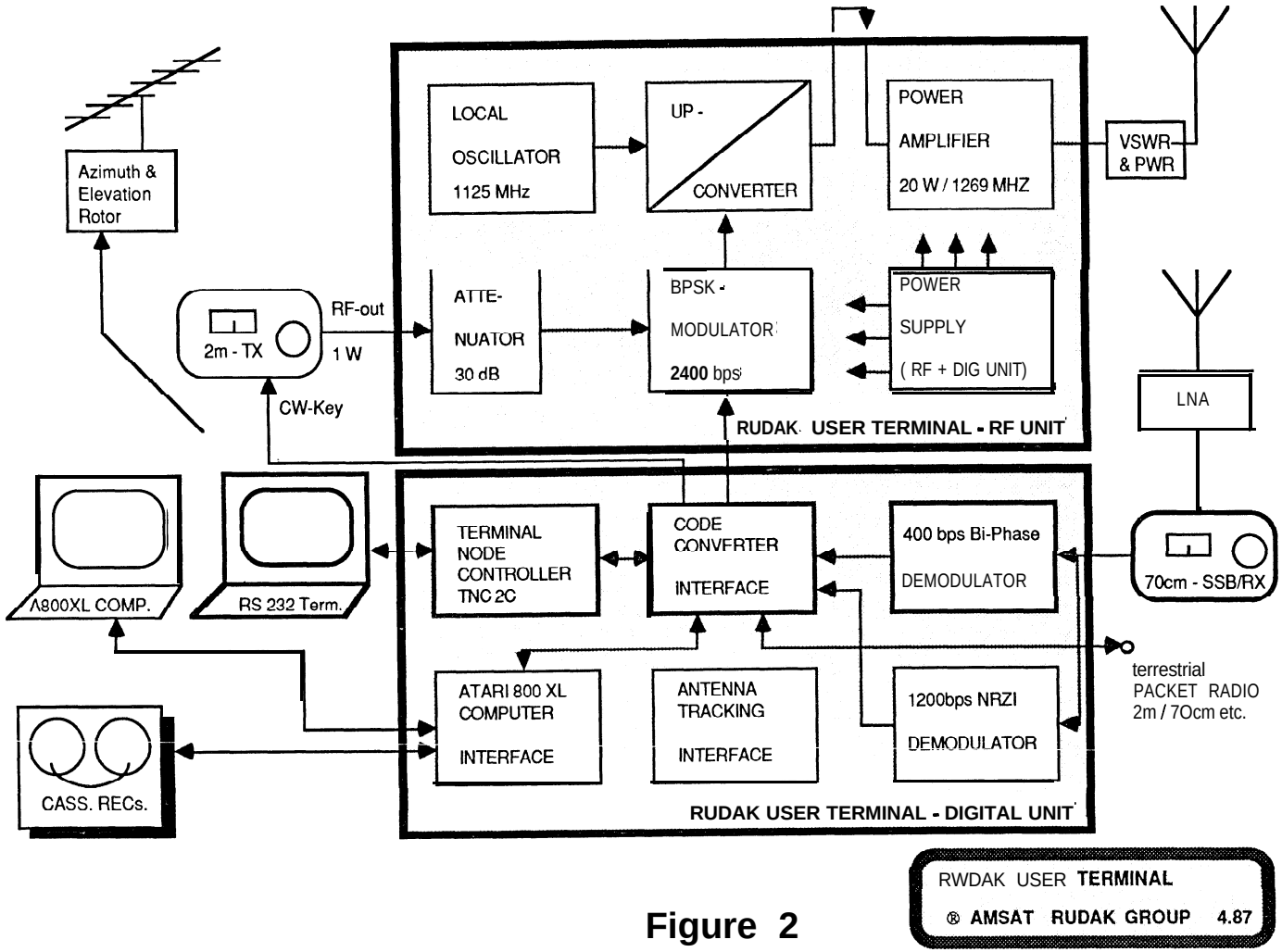

LOCAL
OSCILLATOR
1125 MHz
UP -
CONVERTER
POWER
AMPLIFIER
20 W / 1269 MHZ
VSWR
& PWR
Azimuth &
Elevation
Rotor
2m - TX
RF-out
1 W
CW-Key
ATTE-
NUATOR
30 dB
BPSK -
MODULATOR
2400 bps
POWER
SUPPLY
(RF + DIG UNIT)
RUDAK USER TERMINAL - RF UNIT
LNA
A800XL COMP.
RS 232 Term.
TERMINAL
NODE
CONTROLLER
TNC 2C
CODE
CONVERTER
INTERFACE
400 bps Bi-Phase
DEMODULATOR
70cm - SSB/RX
terrestrial
PACKET RADIO
2m / 70cm etc.
ATARI 800 XL
COMPUTER
INTERFACE
ANTENNA
TRACKING
INTERFACE
1200bps NRZI
DEMODULATOR
CASS. RECs.
RUDAK USER TERMINAL - DIGITAL UNIT
RWDAK USER TERMINAL
® AMSAT RUDAK GROUP 4.87

Figure 2

* Compatibility with FUJI OSCAR 12 formats.
* Built-in Terminal Node Controller (TNC)
* Internal switching for space- or terrestrial packet radio operations.
* AMSAT interface to general purpose computer (ATARI 800XL) for satellite tracking with automated antenna azimuth and elevation control, satellite telemetry decoding with display of measured parameters in engineering units, visibility prediction, data communications in AMSAT block format similar to telemetry blocks of OSCAR10, etc.

To the best of our knowledge, the ATARI 800XL is available in almost any country in the world. This computer has been selected for OSCAR 10 satellite control purposes because of its extremely low price and even more importantly (sometimes) because of its very effective RF-shielding. As soon as another computer is operated close to the sensitive satellite RX, this shielding is appreciated even more.

* Common power supply in the RF-unit.

For the digital unit, we selected a so-called modular design, so that commencing from a common baseline version, the station can be expanded on a step-by-step basis with additional functions, if desired ("matched to the achievable PFD - pecuniary flux density .hi !)

The whole set-up is shown in figure 2. This block diagram shows the required equipments in addition to the user terminal.

Before getting nervous on the very complex configuration shown in that diagram it must be emphasized that it shows the ultimate super dupe version of an amateur satellite home terminal. The minimum required hardware beside the receiver and an RS232 monitor are the 400bps demodulator, the code converter interface and the TNC. This will allow monitoring of RUDAK activities as a looker-on. Doing so and reading hopefully a lot of exotic call signs passing by will definitely motivate to do one more step and participate with active transmissions.

RUDAK Field Test Installation

Since the RUDAK experiment utilizes several newly developed items, in both the hardware and software areas, a comprehensive field test for all components was therefore determined to be mandatory.

The equipment has been installed on top of a 45m high water tower in the city of Ismaning near Munich providing RUDAK experimentation and testing to several amateurs in the Munich region. The block diagram of the installations is given in figure 3.

From here RUDAK is operated exactly the same way as hopefully soon from OSCAR XX in-orbit.

This includes also the testing of the IHU interfaces and the internal communications among the two computers on-board.The equipment configuration of the field test is kept exactly the same as the equipment on-board the satellite so that the achieved performance improvements and hardware changes can also be introduced into the flight hardware.

The RUDAK Software System

The basic operational software (S/W) is available and has been tested. However, it is the software which will hopefully provide a caleidoscope of new features during the in-orbit life time of the experiment.

For the sake of communications among the members of the RUDAK group, the AMSAT control stations and, last but not least, interested experimenters, we defined the S/W in modular way:

The entire S/W system is composed of the subsystems Flight S/W, Ground Support S/W and Test & Simulation S/W.

The subsystem Flight S/W breaks down into so-called tasks such as:

The Task 1.0: ROM operating system (ROS), Task 1 .1: the IPS kernel, Task 1.2 : the AX.25-server, Task 1.3 : the beacon service, Task 1.4 : the traffic control and Task 1.5 : Utilities

The subsystem Ground Support S/W breaks down into:

Task 2.1: RUDAK+IHU Command S/W, Task 2.2 : Integrated S/W package (public domain S/W for ATARI 800XL), Task 2.3 : TNC S/W (i.e. modifications and changes for RUDAK operation of various TNC- S/Ws)

The subsystem Test & Simulation breaks down into:

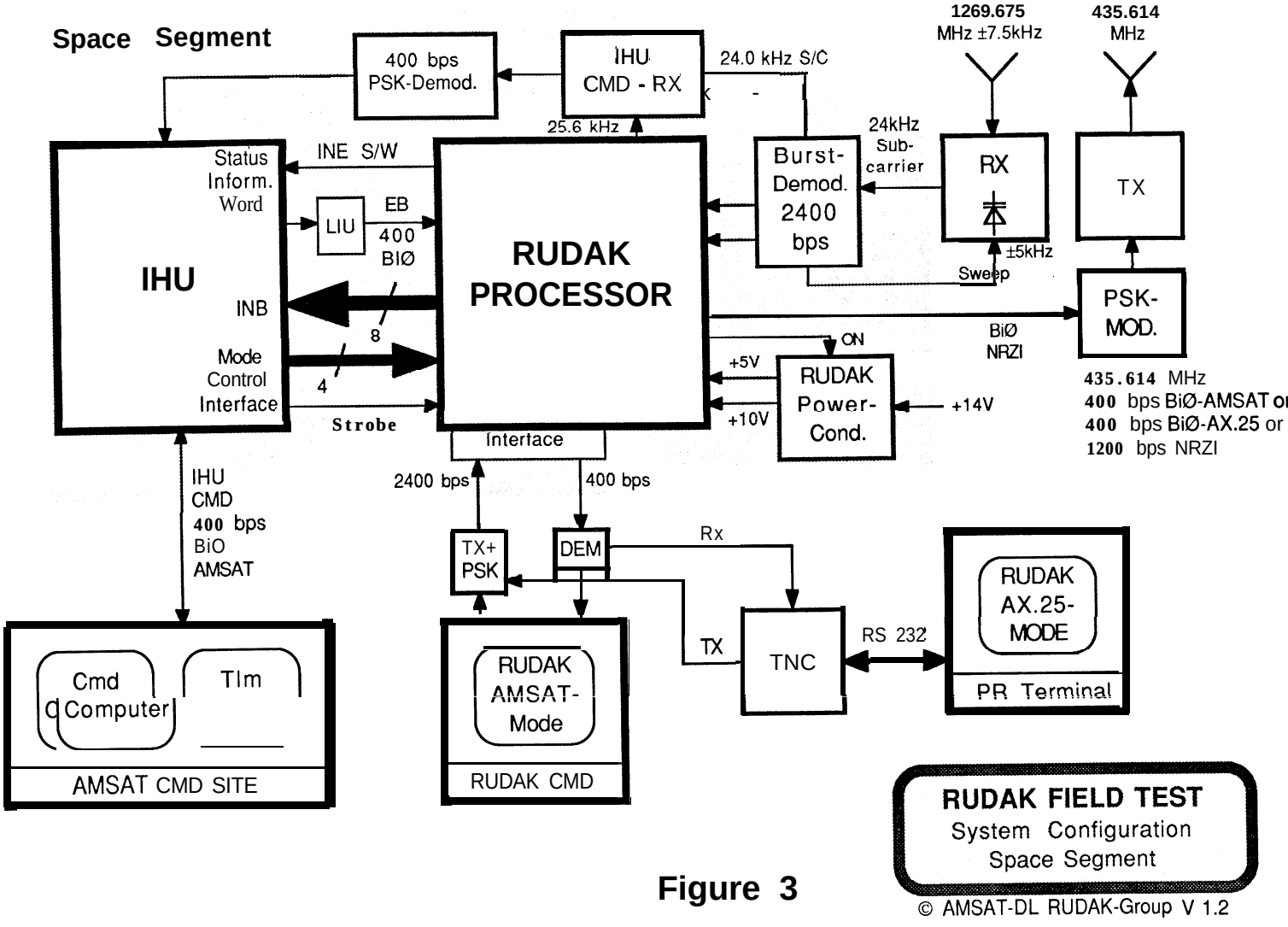

Space Segment
400 bps PSK-Demod.
IHU CMD - RX
24.0 kHz S/C
25.6 kHz
1269.675 MHz ±7.5kHz
435.614 MHz
24kHz Sub-carrier
Burst-Demod. 2400 bps
RX
TX
±5kHz
Sweep
PSK-MOD.
Status Inform. Word
INE S/W
LIU
EB
400
BIØ
IHU
INB
8
Mode Control Interface
4
Strobe
RUDAK PROCESSOR
BiØ NRZI
ON
+5V
+10V
RUDAK Power-Cond.
+14V
435.614 MHz
400 bps BiØ-AMSAT or
400 bps BiØ-AX.25 or
1200 bps NRZI
Interface
2400 bps
400 bps
IHU CMD 400 bps BiO AMSAT
TX+ PSK
DEM
Rx
TX
TNC
RS 232
RUDAK AX.25-MODE
PR Terminal
Cmd Computer
TIm
AMSAT CMD SITE
RUDAK AMSAT-Mode
RUDAK CMD
RUDAK FIELD TEST
System Configuration
Space Segment
© AMSAT-DL RUDAK-Group V 1.2

Figure 3

Task 3.1 Field Test S/W, Task 3.2 In-Orbit Test S/W (Engineering mode)

The RUDAK User Manual

Great care and attention has been devoted to the objective of keeping the user station reasonably easy to built. Therefore the design of the proposed user terminal is such that no special tools or machinery will be necessary to built it. Several components, in particular those of the RF unit have been built and evaluated in many different design approaches testing their suitability for amateur construction.

The results of these investigations have been compiled in the RUDAK User Manual. This manual provides all necessary back ground information on all aspects of the experiment as well as a detailed description of how to built your own terminal.

At the time of writing this article we are in the final compilation and editing phase of the manual. The official release for the complete handbook is presently scheduled for the end of this year, right in time to get prepared for the launch of OSCAR P3C.

Acknowledgements

Many thanks to Dr.Karl Meinzer DJ4ZC and Werner Haas DJ5KQ for their generous support and the phantastic people in the RUDAK team for their excellent and invaluable contributions to this project: Peter Gülzow DB2OS, Stefan Eckardt DL2MDL, Gerhard Metz DG2CV, Knut Brenndörfer DF8CA, Herrmann Hagn DK8CI, Heinz Molleken DL3AH, Don Moe KE6MN/DJ0HC (AMSAT-NA) and Robin Gape G8DQX (AMSAT-UK).

References

[1] Dr.K.Meinzer DJ4ZC, H.Kuhlen DK1 YQ
" Formal definition meeting of the AMSAT RUDAK experiment "
Proceedings of the 4th ARRL Computer & Networking
Conference , San Francisco, USA 1985

[2] K.Brenndörfer DF8CA, G.E. Tucker,WA5NVI
" RUDAK: A status report "
AMSAT QEX 59, Jan. 1987

[3] H.Kuhlen DK1YQ, D.Moe KE6MN/DJ0HC
" The RUDAK Experiment "
73 Magazin, August 1986

Improving Shared Channel Access Techniques for Amateur Packet Radio

Brian Lloyd, WB6R QN

Amateur Packet Radio has come of age. There are now many packet radio users in the Amateur community, well over 20,000 at this point. In the "olden days" when there were few users on a channel at any one time the need for effective channel access techniques was much less than it is now. The problems all result in the same symptoms: poor throughput and lost connections.

Here is a list of the problems:

1. Collisions due to hidden terminals
2. Collisions due to jump-on
3. Unnecessary retransmissions
4. Congestive collapse

The first problem is called the hidden terminal. A hidden terminal is one that shares the channel with your station but cannot hear or be heard by you. If both stations attempt to use a shared resource, for instance a digipeater, the result is numerous collisions at the digipeater due to the failure to wait for the other station to cease transmitting. If you can't hear them you won't wait.

Collisions due to hidden terminals are impossible to eliminate unless you can eliminate the hidden terminals. It is possible to permit all stations to hear all other stations if you improve the radios and antennas used at packet radio stations or you can utilize duplex digipeaters that forward packets in real-time. This permits all users within range of the duplex digipeater to hear when any other station is transmitting (just like we do when we talk on our local voice repeater). This will eliminate hidden terminals.

The second problem, collisions due to jump-on, is caused by the 1-persistent nature (always transmit when the channel becomes clear) of the Carrier Sense Multiple Access (CSMA) software commonly found in our TNCs. If we examine what happens when a station is transmitting and several others become ready we readily see the problem.

1. Station A begins transmitting
2. Station B becomes ready but waits for A to stop sending
3. Station C becomes ready but waits for A to stop sending
4. Station A stops transmitting
5. Stations B and C both "jump-on" and collide

The problem here is that once a station becomes ready to transmit it will, as soon as the channel appears to be clear. The solution here is to reduce the probability that a station will begin transmitting when the channel goes clear when there is more than one station ready to transmit. This technique is known as p-persistent CSMA. The probability for transmission 'p' is varied in order to reduce the chance for collisions without undue effect on throughput. Here is how it works:

1. A station has data to send and waits for the channel to become clear
2. When the channel becomes clear the station generates a random number and compares it with the value of p (the transmission probability). If it "wins" (the random number is less than p) it transmits. If it "loses" it waits for one slot time (a tunable parameter) and repeats this sequence.

This sequence of events guarantees that the station will eventually transmit. The random factor will cause significant variability between the time the channel goes clear and a given station transmits. This provides greatly improved channel sharing.

Let us look at our original scenario and see what happens if the two stations were to use a value of 0.5 for p (50% probability of transmission when the channel goes clear). There are four possible states when the channel goes clear: neither B nor C transmit, B transmits but C does not, C transmits but B does not, both B and C transmit. The result is that the probability of a collision drops from 100% to only 33% and the probability that a station will transmit without a collision is now 66% (after some number of slot times). This is a considerable improvement and will significantly reduce the number of retransmissions required.

Another subject of interest is high incidence of unnecessary retransmissions due to the sending stations failing to wait for a sufficient period of time before retransmitting an apparently lost packet. The firmware that is currently installed in TNCs uses a very simplistic algorithm to set the value of the retransmission timer. This time, known as FRACK, is set to a fixed value, usually 3 seconds. This value is multiplied by n+ 1 where n is the number of digipeaters in the address chain. The result is that the retry timer will be set to 6 seconds when packets are being routed through a single digipeater regardless of all other channel activity.

The problem with this simplistic algorithm is that the retry timer at the sending station may expire while waiting for an ACK when the channel activity is so high that the receiver is unable to respond for a period of time that exceeds the period of the retry timer. The result is that the sending station retransmits the packet when it really should be waiting longer.

There is a solution to this problem; use an adaptive algorithm to adjust the value of the retry timer to suit the channel conditions. In order to make this possible, the stations must collect information about actual round-trip packet time. Each time a packet is transmitted, the sending station should start a timer. When the ACK is received, the timer is stopped. These times should be run through a filtering algorithm to derive a smoothed round trip time (SRTT). The retry timer should then be set to a value greater than or equal to the SRTT (in fact it should be set much longer than the SRTT to accommodate transient peaks in the round trip time). Since the software will arrive at a value for retry time that is based on current channel conditions rather than an arbitrary value, excessive retransmissions will be avoided in the case of heavy channel loading and excessive delays will be avoided in the case of light channel loading. Here is the formula for smoothed round trip time:

$$SRTT' = (1 - \alpha) * RTT + \alpha * SRTT$$

Where:

- $SRTT$ = the previously calculated value for the smoothed round trip time
- $SRTT'$ = new value of $SRTT$
- α = a parameter ranging from 0 to 1; if not adjustable, a suggested fixed value is 0.9
- RTT = round trip time measurement for the current packet

This formula will produce a smoothed calculation of the round trip time. The value of α determines how much effect the new round trip time (RTT) will have on the new value for SRTT ($SRTT'$). Large values of α will give a slow response to changes in RTT while small values of α will give more weight to RTT. In essence, this formula is a low-pass filter for changes in the value of RTT.

So far I have discussed solutions for many problems but I haven't dealt with the problem of "prime time," that period of time in the evening when so many stations are on the air at once that even with p-persistence and a round trip timer, many packets will be lost due to collisions anyway. It is the case with any sort of shared channel that if an excess of traffic is offered, the throughput will drop, leading to more transmissions, more collisions, and even less throughput. This is a downward spiral leading to a malady known as congestive collapse. The only solution to congestive collapse is to reduce the amount of traffic offered to the channel. Round trip timing will help by responding to the delays and slowly increasing the amount of time between retransmissions. There is another, short-term solution known as backoff.

The purpose behind backoff is to have the stations using the channel very rapidly increase the amount of time between retransmissions. In the case of binary exponential backoff, the time between retransmissions is doubled each time a packet

is retransmitted. If our initial retransmission timer is set to ten seconds, the time between first and second retransmissions would be 20 seconds, 40 seconds between second and third retransmissions, 80 seconds between third and fourth retransmissions, and so forth. The result of this is to have the stations retransmitting data rapidly reduce the amount of traffic offered to the channel, permitting the channel to recover and begin moving information again.

The value of all the above suggestions has been proven in the Washington, DC, area. In this area, there are many stations using the KA9Q TCP/IP networking software. The KA9Q TCP/IP networking software in conjunction with the KISS TNC implements p-persistence, round trip timing, and binary exponential backoff. During periods of heavy channel activity when both TCP/IP and "ordinary" TNCs were using the channel, TCP was in all cases able to move traffic when the TNCs were losing connections. The only symptom visible to the operators of the TCP/IP stations was periods of longer delay in the delivery of traffic. Status information garnered from the TCP/IP software showed the round trip time increasing and examination of the transmission characteristics showed that binary exponential backoff was called into use when the traffic was exceptionally heavy.

While this test was not performed scientifically, the results were obvious; TCP/IP, using the above mentioned channel access techniques, delivered traffic when ordinary TNCs were losing connections. A compromise was finally reached when persistence was dropped at the TCP/IP stations to a level below 20% to permit ordinary TNCs priority on the channel. This had no effect on the operation of TCP/IP other than increased delays.

In conclusion it appears that a solution can be found to the problems of collisions due to hidden terminals, collisions due to jump-on, unnecessary retransmissions, and congestive collapse. The problem of hidden terminals can be addressed through the use of duplex digipeaters or guaranteeing that all stations can hear all other stations in a given local area. Collisions due to jump-on can be avoided through the use of p-persistent CSMA. Unnecessary retransmissions can be avoided by the use of round trip timing in the setting of the retransmission timer. Congestive collapse can be avoided through the use of a retransmission backoff algorithm such as binary exponential backoff.

RF spectrum for the amateur radio operator is limited and we must do as much as possible to use what we have more effectively. These techniques will permit us to make the greatest use of whatever shared channel resources are available.

THE NOISE PERFORMANCE OF SEVERAL DATA DEMODULATORS

Hugo Lorente, LU4DXT
Casilla de correo 520
1900 La Plata
ARGENTINA

INTRODUCTION

This paper shows the noise performance of several data demodulators currently being used in packet radio. The bit error rate (BER) in presence of white Gaussian noise was measured on the following demodulators:

1) JAS-1 PSK demodulator [1]
2) Discriminator [2]
3) AMD 7910 modem [3]
4) Differential detector [4]
5) G3RUH PSK demodulator [5]
6) EXAR 2211 demodulator [6]

The noise performance of this demodulators is graphically presented here and a brief description of the test gear used in these measurements is given in appendixes A and B.

METHODS

The test setup for the demodulator's performance measurement is shown in figure 1. The V-52 generator produces a pseudo-random test sequence as recommended by the CCITT V-52 standard [7]. Details of this generator are given in appendix A.

The FSK modulator is an EXAR 2206 integrated circuit,

The PSK modulator was specially built for these tests using digital techniques: clock, binary counter, sine look-up table in EPROM and D/A converter. The carrier frequency was set to 1600 Hz and the PSK demodulators were tuned for operation with this frequency,

For both, the FSK and the PSK modulators, an output peak amplitude of 1 volt was chosen. The JAS-1 demodulator was modified to work correctly with this amplitude (the 10-kΩ resistance at the input of U2 was changed to 100-kΩ).

The noise generator was built, as shown in appendix B, filtering a pseudo-random sequence (PN) with a low pass filter [8][9]. This a very simple way to obtain Gaussian noise with a well known power spectral density.

The outputs of the modulator and the noise generator are added by the operational amplifier A1 as shown in figure 1. The output of the noise generator was adjusted to obtain the needed signal to noise ratio. This ratio is defined as:

$$S/N = \frac{Eb}{\eta} = \frac{a^2 \cdot Tb}{2 \cdot \eta} = \frac{a^2}{2 \cdot \eta \cdot Fb} \quad \text{Ec 1}$$

Where Eb is the mean bit energy, η is the one-sided power spectral density of the noise. a is the modulators' output peak amplitud (1 volt), Tb is the bit duration and Fb is the bit rate (1200 bits/s).

The delay shown in figure 1 is another demodulator, identical to that being tested, but this one demodulates the signal in absence of noise and in that way provides a no error data stream with the demodulation delays. The output of this demodulator is also used to recover the bit clock using a state machine [10].

The demodulators' outputs are compared in an exclusive-or gate whose output is a logical "one" only when its inputs are different, i.e. : when there is an error. This errors are sampled by the recuperated bit clock and counted with a standard digital counter.

With this test setup the errors were counted together with the measurement of the duration of the test. At high error rates large error counts are accumulated in a short time, but at low error rates the time interval needed to accumulate even a few errors becomes very large. A large error count is needed to obtain meaningful data, so that it was necessary to adopt the following trade off:

1) Accumulate if possible a large number of errors (N>1000).
2) At low error rates accumulate at least 100 errors,

RESULTS

With the duration of the test and the bit error count, the bit error rate was calculated as:

$$BER = \frac{N}{Fb * T} \quad \text{Ec 2}$$

Where N is the bit error count, Fb is the bit rate (1200 bit/s) and T is the duration of the test.

The bit error rate was plotted versus the signal to noise ratio as shown in figure 2 and the signal to noise ratio needed for a BER = 1E-04 is indicated in table 1.

The JAS-1 demodulator is the best performer and runs very close to the ideal demodulator. This is very important in satellite work using SSB reception where one dB gained at the demodulator is equivalent to one dB gained at the antenna.

It's also interesting to note the poor performance of the most popular packet radio FSK demodulator: the EXAR 2211.

APPENDIX A

The test sequence generator circuit is shown in figure 3. It was built using a 9 stages shift register (IC1 and IC2) whose outputs Q4 and Q8 are added together in a module-two adder (IC3A) and the result is fed back to the first stage, The gates ICY, IC5A and IC5B form the start-up circuit to avoid the "all ceros" sequence. The generator's 1200 Hz clock is built around the Integrated circuits IC6, IC7A and IC7B.

This test sequence generator is a very useful l tool when working with data communication equipment; its bit rate is easily changed choosing the right IC6 output and/or selecting the adecuate crystal resonator.

APPENDIX B

The noise generator is shown in figure 4. The 24 stages PN sequence generator is formed by the shift registers ICI, IC2 and IC3 whose outputs Q0, Q1, Q6 and Q23 are added together in a module-two adder (IC4) and the result is fed back to the first stage. The IC5C, IC5D and IC5E inverters form the start-up circuit to inhibit the "all ceros" sequence, The inverters IC5A and IC5B are part of the 100KHz crystal controlled clock. The operational amplifiers IC6A and IC6B are part of a three pole Butterworth filter with a cut-off frequency of 3400Hz. The output noise level is adjusted with a precision IO turns potentiometer with a digital dial; it is easier and more precise to trust in the potentiometer dial's indication than trying to measure the generator's noise power output. Finally the operational amplifier IC7 buffers the noise output.,

A FN sequence generated with a circuit like the one shown here has a low frequency one-sided power spectral density η given by:

$$\eta = K^2 * \frac{V^2}{2} * \frac{1}{Fc} \quad \text{.....Ec 3}$$

Where K is the reading of the potenciometer's dial ($0 <= K <= 1$), V is the peak to peak amplitude of the PN sequence (in this case the CMOS logic output swings between cero and Vcc, that is V = 15 volts) and Fc is the clock frequency (100KHz). Replacing in Ec 3 we have:

$$\eta = K^2 * \frac{15^2}{2} * \frac{1}{1E+05} =$$

$$= 112.5E\text{-}05 * K^2 [\ V^2/Hz\ ] \quad \text{.....Ec 4}$$

REFERENCES

[1] "A PSK Demodulator for the JAS-1 Satel-1 ite", F. Yamashita, JSIUKR, QEX, August 1986, pp 3-7.

[2] "Optimun Binary FM Reception Using Discriminator Detection and FI Shaping", A.A. Meyerhoff and. W.M. Mazer, RCA Review. December 1961, pp 698-728,

[3] "Am7910 FSK Modern Wor ld-Chip", Advanced Micro Devices data sheet,

[4] "Differential Detection of Binary FM", R.R. Anderson, W.R. Bennet, J.R. Davey and J , Salz, Be l l System Technical Journal, January 1965, pp 111-159.

[5] "JAS-1 Modem", J. Mi l ler, G3RUH, Proceedings of the 4th Annual AMSAT Space Symposium, Dallas, Texas, November 1986, p 148.

[6] "XR2211 Demodulator/Tone Decoder", Exar Integrated Systems data sheet.

[7] 'Characteristics of Distortion and Error-Rate Measuring Apparatus for Data Transmission", International Telegraph and Telephone Consultative Committee (CCITT), Mar del Plata, 1968.

[8] "Shift Register Sequences", S.W. Golomb, Ho lden-Day, Inc., San Francisco, CA, 1967.

[9] "Some Statistical Properties of Smoothed Maximal Length Linear Binary Sequences", P.D. Roberts and R.H. Davies, Proc. 7.E.E, 1966, vol. 113, p 190.

[10] "Xerox Packet Interfase Board", J. Bloom, KE3Z, et al., AMRAD Newsletter, Nov./Dec. 1984.

TRADEMARKS

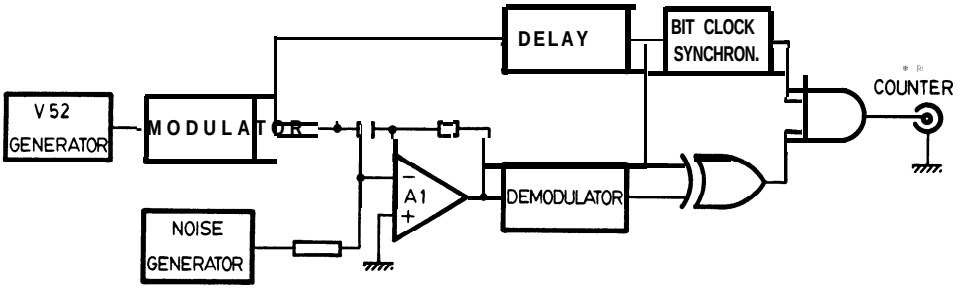


FIGURE 1

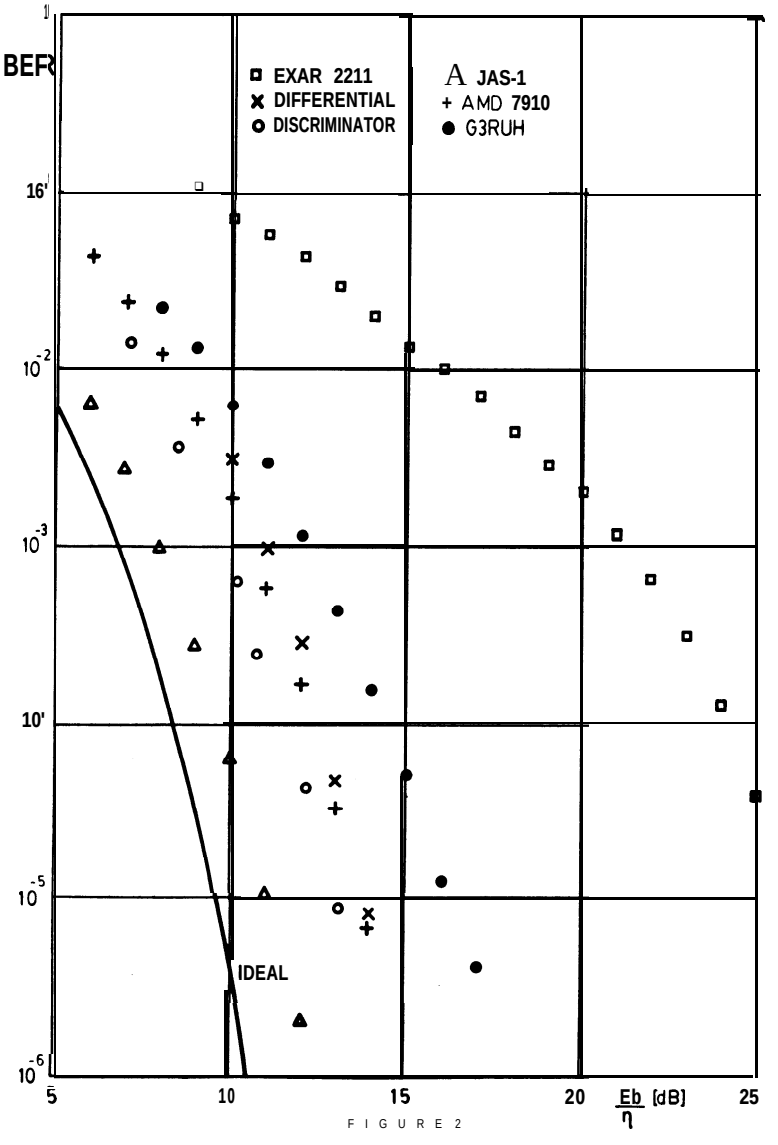


FIGURE 2

DEMODUL.	JAS-1	DISCR.	7910	DIFF.	G3RUH	2211
S/N FOR BER=1E-04	9.7 dB	11.5 dB	12.3 dB	12.6 dB	14.4 dB	24.2 dB

T A B L A No 1

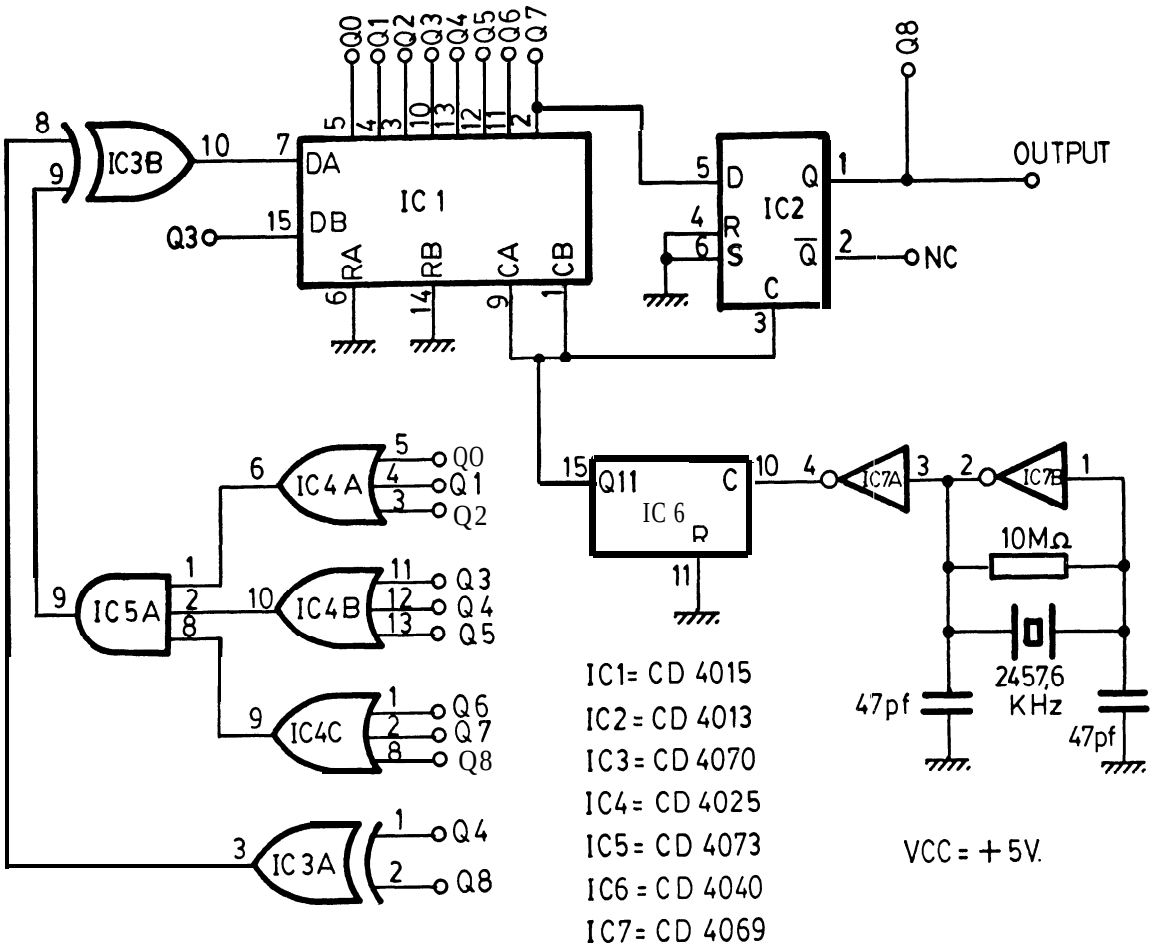
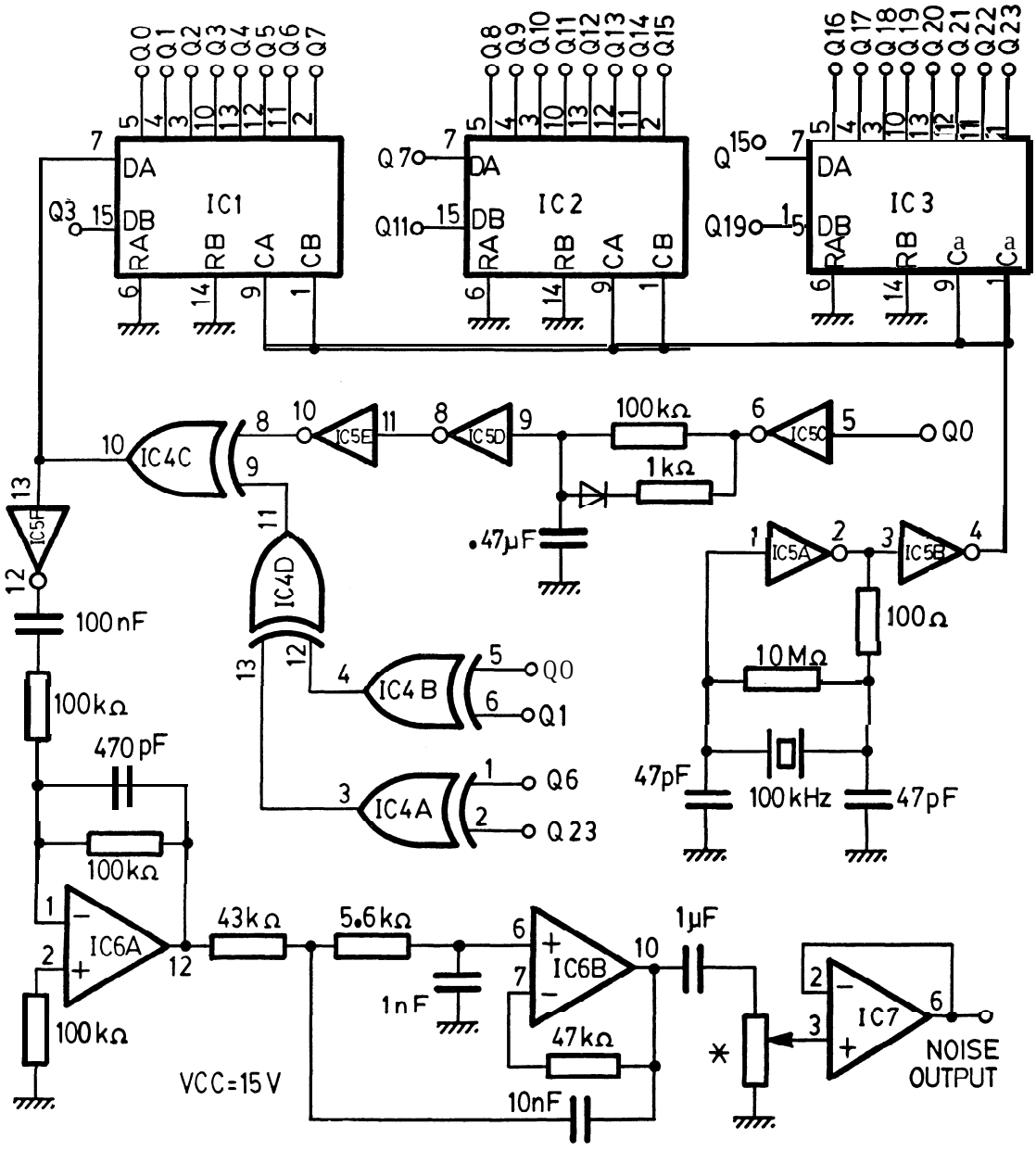


FIGURE 3



IC1-IC2-IC3 = CD 4015
IC4 = CD 4070
IC5 = CD 4069
IC6 = LM 747
IC7 = LM 741

* TRIMPOT 3610-S
10 k OHMS-10 TURN
POTENTIOMETER
WITH DIGITAL DIAL.

FIGURE 4

Overview of the TEXNET datagram protocol

T.C. McDermott, N5EG
Texas Packet Radio Society, Inc.
PO Box 831566
Richardson, Texas 75083

Abstract

This article is an overview of the TEXNET datagram network protocol. It is not intended as a specification. It deals with the protocol within the network (intra-network). It does not deal with layer 3 protocol issues between the user and the network (inter-network).

Introduction

The actual data-transfer and routing portion of a network protocol is the simpler aspect of the design. More difficult problems are encountered in administration and maintenance of a network. The majority of the effort in developing TEXNET centered on solving these network administration and initialization problems.

Definitions

These definitions are important:

USER - a user of the network. Normally thisuser utilizes a 2-meter radio and a TNC.

NODE - a TENXET network node containing 2-meter 1200-baud and 450 9600-baud radios, modems, and a Node-Control-Processor (NCP)

PHYSICAL CHANNEL - a serial channel implemented in hardware. For example: the 450 9600-baud radio is connected to a physical channel, the 2-meter 1200-baud radio is connected to a different physical channel. Each node has at least 1 physical channel, and may have up to 3 physical channels.

LOGICAL CHANNEL - An entry in a table defining an Ax.25 connection. Each physicai channel may have several logical channels active simultaneously. A logical channel may be either a user channel or a network-trunk channel. Each physical channel may have any number of logical channels, but a software-defined maximum of 20 is the practical limit.

LOGICAL CHANNEL NUMBER - a l-byte dynamic number associated with a particular AX.25 connection at a node. As connections come and go at a node, the numbers will be reassigned. May be either a network trunk or a user connection.

NODE NUMBER - a l-byte static number that defines a given node. The correlation between ASCII node names and node numbers is determined by the nework administrator when the EPROM is programmed for a given network node.

Protocol basics

TEXNET is a datagram-based protocol. BY this is meant that the two terminating nodes (the nodes at the entry and exit points of a particular circuit) maintain tables that specify how each particular user is connected. Intermediate nodes within the network do not have knowledge of any users (except those users that terminate at that intermediate node). The purpose of an intermediate node is to perform transit-routing. Thus each node must contain a data-blase that represents the entire network. Initializing and developing this database is a challenging task.

Layer 2

The ARRL specification "AX.25" is the standard for all layer 2 connections. There are two types of L2 circuits, user circuits, which are either version 1 or version 2, with reserved bits unused, and network trunk circuits, which are always version 2, but which are distinguished from user trunks by having reserved bits 4 and 5 in the AX.25 header (which are normally set to one when inactive) used. Bit 4 is used to indicate user (inactive, or 1) versus

network (active, or 0). It is required to use the bits in the header field because the PID field, which would normally be used for this function, does not arrive until an I-frame is sent, and indication of status is required prior to this time. Bit 5 is used for disconnect timer control.

The T3 time-out timer of AX.25 is redefined as follows:

Bit 5 = 1 (inactive) = T3 is the inactivity disconnect timer. No traffic on the logical channel for T3 seconds causes a disconnect.

Bit 5 = 0 (active) = T3 is the link keep-alive timer. This is the normal Ax.25 version 2 definition. Network trunks utilize this to provide the permanent virtual circuit function. This bit could also be utilized by a user to inhibit disconnect timing for trouble-shooting purposes. Normally the user should not utilize this function.

These bits were intentionally made hard-to-reach for 'normal' users so they would not be tempted to activate them.

Adherence to the 128-octet frame length limit is maintained. TEXNET will perform packet fragmentation if necessary to comply with this limit. This is done in order to maximize the number of frame-buffers available for processing. A large pool of frame buffers can enhance network performance under certain circumstances.

The sender and receiver of all packets is identified by the amateur-callsign of the node(s) and or user involved, per FCC requirements. No callsign reassignment is performed. TEXNET utilizes the *** LINKED message to cause bulletin boards to work properly.

Laver 3

The TEXNET layer 3 protocol consists of 5 bytes that head every network-trunk packet. Additionally, certain network administration packets may contain further information in the 'data' field which follows the network header. All network packets are sent as I-frames, with the exception of one network initialization command, which is sent utilizing the UI-frame.

Each network packet can be viewed as a series of bytes, starting with the first data byte within the I-frame. In order to minimize the number of data bytes of overhead, each node within the network is assigned a unique l-byte value as it's node number. Further, each user connection is assigned, dynamically, a l-byte value at each node. Thus 2-bytes specify both the node and the exact user connection involved. This allows 4-bytes to specify both ends of the connection. Information about both ends is included in order to allow error-recovery by intermediate nodes.

Network header block (NHB)

1st byte	NHB.RNN	= remote node number
	NHB.RLC	= remote logical channel number
	NHB.LNN	- local node number
	NHB.LLC	= local logical channel number
5th byte	NHB.NCF	- network control field
6-nth	data	= user data or network data (determined by control field)

The valid range of l-byte numbers is l-255. Zero is reserved to indicate an invalid node or channel.

The following list specifies the currently defined control field values. Values in the range O-127 are commands, and those in the range 128-255 are error messages (returned to the user as Network Information Code, NIC, plus the error number).

Decimal value	Definition
0 = NODEIN	Node initialization
1 = STATRQ	Current statistics request
2 - STATRS	Statistics response
3 - NODEID	Node identification + real time clock and date initialization
4 = POINTQ	Point update
5 - POINTR	Point response to update
6 = YSTATR	Yesterday's statistics request
16 = CONREQ	Circuit connect request
17 = CONACK	Circuit connect acknowledge
18 = DISREQ	Circuit disconnect request
19 = FLOWON	Set logical channel to not busy
20 = FLOWOF	Set logical channel to busy
21 = USERIN	Contains user data
32 = MYNAME	Distant node telling us his ASCII name
33 = TIMEUP	Update remote node's real time clock and date manually

Certain of the above commands have additional data included within the data field after the first 5 header bytes. These are defined subsequently.

Control fields of 48 to 63 are broadcast-type packets. Broadcast packets are sent to the entire network via controlled 'flooding'. Each broadcast type datagram has a broadcast stack following the 5 header bytes.

48 - ALRTON	Alert status ON
49 - ALRTOF	Alert status OFF
50 = ROUTLD	Delete a node from the routing table
51 = IMHERE	New node announcing it's presence to the entire network

Network circuits

A network circuit is a communication path between two users via a network. In TEXNET, only the two terminating nodes have knowledge of the user circuit. These two terminating nodes communicate with one another to set-up, maintain, and tear-down the circuit.

Circuit set-up/teardown

Control fields 16-21 are the commands necessary to establish and delete user connections through the network. Each terminating node maintains several flags about the state of each user logical channel. These flags, and the command fields regulate the connection. The flags are:

1 - user is in command mode
2 - incoming connection request
3 - outgoing connection request
4 - circuit is in information transfer mode
5 - local logical channel is busy
6 - remote logical channel is busy
7 - remote user is an authorized administrator

The terminating nodes exchange the fields 16-21 to control both set-up/teardown, and end-to-end flow control. Flow control is activated between user logical channels at the respective terminating nodes. AX.25 busy is entered when either local or remote busy is encountered on a circuit. This prevents over-filling the network frame buffers within each network node when the sending user is transmitting faster than the receiving user can accomodate. It does not flow-off other users at those nodes.

Connect request data field

The connect request data field contains the callsign of the requesting user, the callsign of the desired target user, and the digipeater string (if any) to be utilized at the remote network node.

Packet fragmentation

Since the maximum packet length is 128 bytes, and the network header takes 5 bytes, if the incoming user data packet is between 123-128 bytes in length, it is fragmented into two packets. The packet IS NOT re-constructed at the terminating node, but rather is sent as two packets to the remote user.

Network initialization

TEXNET is a network with no fixed routing assignments. This current version is a significant improvement over the previous version where the entire static network map was stored within the EPROM at every node. This necessitated changing every EPROM in the network whenever the topology changed. The automatic routing system developed is described below.

The purpose of the automatic routing system is four-fold::

1. Notify all possible neighbors whenever a new node is turned on - even if we do not know who those neighbors are.

2. Initiate AX.25 trunk connections between all neighbors, and associate each logical channel with each neighbor.

3. The new node must 'announce' it's presence to the entire network.

4. Routing table entries between the new node and every other node in the network must be generated. Furthermore, the optimum path must be selected for use as the primary route (where two or more routes exist), and an alternate route must be selected, to be utilized if the primary route fails, assuming that there is redundancy in the paths. Further, it is possible the paths on several different physical channels (different radios) may exist simultaneously, therefore priority must be resolved.

UI - frames

The UI-frames are used by a new node (one that has just completed a processor reset sequence) to alert it's presence to all nodes within hearing range. These frames are an 'invitation' to a neighbor node to start a network trunk back to this new node. The UI-frame is sent 'retry' number of times, to assure that all other network nodes within hearing distance will have a good chance of receiving it. This is transmitted on all physical channels unless the node database has a particular physical channel disabled from network trunk initialization.

The format of this UI-frame is defined as follows:

PID - 0C3 Hex
1st byte - Our network number
2nd - Our node number
3-n bytes - Our neighbor exclusion table

'Our network number' is a l-byte 'network identifier'. This allows two (or more) different TEXNET networks to co-exist on the same frequency without logical interconnection. This may be useful for network testing. Additionally, this can be very useful if it is desired to interconnect two TEXNET networks with a gateway station. The gateway can communicate to both networks on the **same** frequency, but the two networks will not logically interconnect. This would be

useful, for example, in constructing a single satellite gateway to two different geographically adjacent networks. 'Our node number' is our assigned l-byte number, and the neighbor exclusion table is a listing of those other nodes which we disallow direct network trunk connections to. This is useful for locking certain paths out (restricting the network topology) during network testing, or eliminating known marginal paths from trunk initialization. Receipt of this UI-frame will cause the receiving node to 'start' (SABM) an AX.25 circuit back to the sender of this UI-frame.

Node identification command

When a connection trunk start is received, the receiving node sends the NODEID network command. This causes the receiving circuit to record this nodenumber into the logical channel table so that association between a logical channel and a particular neighbor node number is made.

Additionally, the NODEID contains the real-time clock and date from the node that received the UI-frame. This automatically keeps new (and restarted) network nodes operating at the correct time and date.

Routing path initialization

After the network trunks have been started between nodes, the node must announce it's presence to the network. It must convey to the network how it is connected to it's neighbors (what it's connectivity to the network is), it must convey it's ASCII node name, and it must convey it's one-byte node number.

The new node will now send the 'IMHERE' I-frame to all of it's neighbors that have started an AX.25 connection to it. Since this is a broacast type of packet, it will be sent to the entire network. Each node in the network will return the 'MYNAME' packet back to this node.

The 'IMHERE' packet data field (in fact, all broadcast packets) contains the following information:

0-4th byte: Normal 5-byte TEXNET header

5th byte: RTHOPS - route hop counter (incremented at each node while building the routing table)

6-20th bytes: RTSKDP - 'push-down stack' holding the previous transit node #s

21-27 byte: RTNONA - ASCII name of the subject node

The hop-counter is incremented by each node that re-broadcasts the packet. Each node also pushes its own node number onto the stack. If the stack contains our node number, then we discard the broadcast packet, since we've already seen it. If the broadcast packet does not contain our node number, we add it to the stack, increment the hop counter, and send this packet to all of our neighbors. This is how infinite routing loops are prevented in TEXNET.

Network response to initiating node

Each node that receives the broadcast 'IMHERE' packet adds the station to it's routing table, if it is not already there. If this station is already there, then the routing table entry is replaced with the new route only if the hop count is less than the current hop count in the table.

At this point, the node adds a physical channel 'hurdle' to the hop count. Routing received on a high-speed network trunk is preferentially treated. That is, if our node has both 9600-baud and 1200-baud trunks, and it receives routing information on each, the 1200-baud trunk will have an additional number of hops arbitrarily added to the hop count before the entry is stored in the routing table. This gives preference to the 9600-baud path, but still allows alternate routing via another physical channel. Also added to the routing table are the ASCII nodename, and it's nodenumber, and the logical and physical channel number that the route was received on (which becomes the 'return route' to that remote node).

The actual algorithm is somewhat more sophisticated. It tests for the case of a previous network node restarting, and prevents duplicate primary and secondary entries under these and other conditions. Receipt of an improved route causes it to become the new primary route, and the old primary to become the new secondary route.

Routing

Each routing table entry contains the primary and secondary (if secondary exists) routes to a particular node. This table indexes the desired destination of the packet to a particular logical and physical channel that leads to that destination. When a packet is received, the field 'REMOTE NODE NUMBER' is examined. If this field is not equal to our node number, then the packet must be transit routed toward the destination. The routing table supplies the information on which physical and logical channel the packet should be sent. No data is altered within the packet.

If however, RNN = our node number, then the packet is for our node. We do not forward the packet, but rather parse the control field, and act appropriately.

Network administration

A number of network commands are used for administration (such as the IMHERE and MYNAME types). These packets are exchanged between nodes, and are independent datagrams between nodes. They are defined as either command with required response, command without required response, or as response, as follows:

TYPE	COMMAND W/RESP	COMMAND WO/RESP	RESPONSE
0 - NODEIN		X	
1 - STATRQ	X		
2 - STATRS			X
3 - NODEID		X	
4 - POINTQ	X		
5 - POINTR			X
6 - YSTATR	X		
16 - CONREQ	X		
17 - CONACK			X
18 - DISREQ		X	
19 - FLOWON		X	
20 - FLOWOF		X	
21 - USERIN	this is user information		
32 - MYNAME			X
33 - TIMEUP		X	
48 - ALRTON		X	
49 - ALRTOF		X	
50 - ROUTDL		X	
51 - IMHERE	X		

Identification

Periodically, the network will transmit a UI-frame with the normal layer 2 (PID = FO) PID field. This packet consists of the layer 2 amateur callsign of the node in the SENDER field, the text 'ID' in the RECEIVER field, and the ASCII name of the node in the data field. This identifies the node per FCC requirements.

Response fields

A simple algorithm is used to determine how to construct the response datagram to any command. The Network Header block bytes are reversed. This causes the local and remote node numbers to change place, and the local and remote logical channel numbers to change place, then the appropriate control code is inserted. In this way, the response is returned to the correct requestor.

Response field data

Each response field may have response data associated with it. The following table lists the response data.

TYPE DATA FIELD CONTENTS

2 - STATRS Compressed 21-byte binary statistics block. This block is expanded into English by the terminating node when delivered to the user.

5 - POINTR Binary block containing the status value of the input-point word (up to 13 input points may be monitored).

Error recovers

In case an error is detected by an intermediate or terminating node, that node performs a similar network header block reversal, but then overwrites it's node number into the local field, and 0 into the logical channel field, and adds the appropriate Network Information Code (NIC) - the error code. In this way the user is informed about the error, and the name of the intermediate node that detected the error. Control fields 128-255 are reserved for the various NIC codes.

Isolation of errors to a particular node is very useful in fault sectionalization. That is, identifying either a particular defective node, or a particular marginal trunk path.

Transport layer (layer 4)

No transport layer protocol is included within the terminating nodes, or within the TEXNET definition. Consider the following scenario: User 1 is sucessfully transfering data through the network to user 2. Channel congestion on the user 1 to TEXNET frequency (the 2-meter user input channel) causes the user 1 to TEXNET link to reset (SABM), possibly resulting in the loss of data frames. A transport function entirely within the network could not correct such loss of frames, and could not correct a complete loss of connection to the network. The connection would either proceed with data errors, or would terminate abnormally.

Instead, the IS0 reference model defines that the transport layer is conducted between the end-users. In this way loss of data or of connection anywhere within the communication path is detected and corrected by the layer 4 (see, for example CCITT X.224 class 1). The IS0 reference model assures that end-to-end data integrity is maintained.

Practical experience with TEXNET has shown that we have not had a need for a transport-like function inside the network to improve the quality of network service. All known instances of data errors have occured between the individual users and the network, not within the network. We have instead devoted limited EPROM space to additional features, such as the network bulletin board, the conference bridge, and the weather-service interface, which are heavily used by our community.

Author's opinion

The definition and adoption of an inter-network, or user-to-network standard (layer 3)within the amateur community is of critical importance. The definition of the intra-network protocol (such as this document) is of secondary concern. It should not matter to any user whether the underlying network is TEXNET or any other network, the interface should be the same regardless. Future developments, such as transport layers, and more, are critically. dependent upon this standard. Ideally this standard would encompass networks, bulletin boards, and future services.

Use of tranport-like functions within the network can improve the quality of service delivered by the network if needed, but do not assure end-to-end integrity. Reference to transport, or layer 4 should be restricted to a protocol which is end-user to end-user in nature.

CSMA MULTIHOP NETWORKS
Throughput Analysis

by Dr. Bob W. McGwier N4HY

Amateur Radio uses packet radio in a broadcast mode and typically uses omnidirectional antennas for transmission. If the two ends of the data transfer are not within communication range, we will use a digipeater. The protocol we use for medium access control is carrier sensed multiple access (CSMA) protocol. Under CSMA, a terminal will not transmit if it hears a transmission in its neighborhood. In a multihop environment (digipeater) such as ours, CSMA is subject to "hidden transmitter" interference. This is when two or more transmitters outside hearing range of each other key up at the same time and interfere with one or more recipients of the two or more packets being simultaneously transmitted. We call this a collision. We could almost completely eliminate this problem with busy tone multiple access (BTMA) protocol. The "almost completely" in the previous sentence is caused by the finite speed of light and the response time of the receiver bringing up the busy tone. BTMA has implicit in its nature seperate transmit and receive operation. Several papers have appeared analyzing throughput in CSMA packet radio networks. The best papers and the most useful (not always the same) are by Boorstyn and Kershenbaum and we include one of their papers as our only reference as it contains the most complete set of references and is the basis for the work described here. They have introduced a continuous time Markov Chain model which lends itself to numerical techniques for finding the steady state response of moderate sized (100-200 nodes) networks. This type of model allows for dependencies between non-adjacent nodes to be modeled and analyzed. The primary purpose of this paper is to introduce these ideas to the amateur radio literature, to supply some rigor to Clark's "But Wait There's More" treatise, and hopefully give one more nudge to do something about the problem.

Our Packet Networks and CSMA

Our networks are made up of terminal node controllers (hereinafter TNC or simply node) with radios for broadcasting packetized data over limited distances (forget HF, that problem is enormously complicated). In the most general case, the source and destination nodes will not be able to hear each other directly, and will digipeat through intermediate TNC's. We assume that the network topology and traffic requirements operate on a time scale sufficiently long to establish steady state conditions. We assume an idealized AX25L2V2 model and completely neglect COSI, IP, NET/ROM, or TEXNET and any point by point acks or dynamic routing these systems may employ.

The Packet) Radio Markov Model

We make a restrictive model, which we hope will include only the effects of CSMA and not the 2211 modems we all use. Our model is

1) Nodes schedule transmissions to neighbors, that is they can hear each other according to a independent Poisson point processes. The arrival times of crashes on 75 meters on a summer night is modeled by a Poisson point process quite well.

2) Packet lengths are exponentially distributed and are generated independently at each transmission. This can be greatly relaxed but we will not do so here.

3) The propagation delay between adjacent nodes is zero.

4) Under CSMA, we will begin our scheduled transmission if we are neither sending or receiving or hearing a packet.

5) Nodes receive with perfect capture. A bad assumption that allows us to do the mathematics that follows. This means if A transmits to B and then during this transmission a neighbor of B, node C, begins to transmit, node B still receives A's transmission. Thus the collision only happens if the recipient of C's packet can hear the A node transmission. In this case, C must retransmit.

6) Links are error free (2211's and MF-10's and OUR radios????). Again this assumption is to allow ease of mathematical modeling and will result in an over-estimate of the throughput.

7) Acks always make it and in zero time. (Chuckle Chuckle). The obvious inadequacies of this model are clearly being VERY generous to our physical links and are designed to "single out" the CSMA component of the throughput of our networks.

CSMA-Markov Chain Analysis

Let s_{ij} be the desired rate of transmission from Node i to Node j. Let i be a node, N_i all the neighbors of i and N_i^* be N_i and i. Let g_{ij} be the scheduled packet rates. In order to achieve s_{ij}, we must have $g_{ij} \geq s_{ij}$. Let $P(A)$ be the probability that all the nodes in a set A are not transmitting at a random instant (idle). In the steady state, $P(A)$ represents a time average. A scheduled packet from i to j will be sucessful if it finds the system in a state whereby both neighborhoods of i and j are idle. Therefore, the important quantity in determining the throughput will be $P(A)$. Since Poisson arrivals see time averages,

$$\frac{s_{ij}}{g_{ij}} = P(N_j \cup N_i) \tag{1}$$

Let g_i be the total shceduling rate out of node i and let $1/\mu_i$ be the average length of packets generated by i.

$$g_i = \sum_{j \in N_i} g_{ij} \tag{2}$$

The Markov model we choose has as its state the set of nodes transmitting. Let D be this active or busy set. D^c is the idle node set. The set D provides enough information about legal transitions which are either completions of a transmission with exponential rates μ_i for $i \in D$ or new transmission rates g_j for $j \notin D$ and $j \notin N_i$, $i \in D$. The latter condition results from the CSMA zero propagation delay assumption.

CSMA forces the nodes in D to be unconnected. Let N_D denote the set of all neighbors of nodes in D and D itself. Let $D + j$, $D - i$ be independent or unconnected sets formed by adding or subtracting a node from D. Assume that the network is stable for rates g_i and μ_i. Then the steady-state probabilities $Q(D)$ of states D follow the global balance equations

$$\left(\sum_{j \in D} \mu_i + \sum_{j \notin N_D} g_j\right) Q(D) = \sum_{i \in D} g_i Q(D - i) + \sum_{j \notin n_d} Q(d + j) \tag{3}$$

It follows that

$$Q(d) = \left(\pi_{i \in d} \frac{g_i}{\mu_i}\right) Q(\phi) \tag{4}$$

$Q(\phi)$ is the probability that all nodes are idle. Normalizing this to unit mass we get

$$Q(\phi) = \left[\sum_{\text{all } D} \prod_{i \in D} \frac{g_i}{\mu_i}\right]^{-1} \tag{5}$$

For this Markov Chain to have a steady state the null transmitter state must be positive recurrent $Q(\phi) > 0$. Finding all the independent sets D for equation 5 is an NP-complete problem. There is a very clever algorithm which allows the special structure of this formulation to be applied to networks of the size already mentioned. To find the throughput, we need the probabilities of being idle $P(A)$. Clearly this is found by summing Q(D) over all D that are contained completely in A" the compliment of A.

$$P(A) = \sum_{D \subset A^c} Q(D) \tag{6}$$

Let

$$SP(B) = \sum_{D \subset B} \left(\prod_{i \in D} g_i/\mu_i\right), \quad SP(\phi) \equiv 1. \tag{7}$$

$$P(A) = SP(A^c)/SP(V) \tag{8}$$

where V is the set of all nodes. Also

$$\frac{s_{ij}}{g_{ij}} = \frac{SP([N_i \cup N_j]^c)}{SP(V)}, j \in N_i. \tag{9}$$

We solve (9) iteratively for the g_{ij} and the throughput is identified in the steady state under this model. An analysis of the very simple network with four nodes in a linear topology is accomplished in several papers found in the bibliography in [1]. We used it to check out our program. The results are somewhat astounding. Given the four node linear topology with our convenient assumptions we get that the max throughput in the network occurs when the scheduling rate divided by the mean packet length g_1/μ_1=0.71. This maximum throughput is 0.128 packets per second. In a star topology with four legs and only five nodes in each leg the mean maximum one-way throughput was *0.054* packets per second. In a ring with 5 nodes the max throughput is 0.100 packets per second. As my final run, I took a map of EASTNET done by Zwirko, K1HTV, and under generous assumptions let the rate s_{ij} correspond to 1200 bps and assumed the mean packet length was 80 characters (a line). Using the 40 most important sites (subjective opinion) and going by word of mouth as to who had solid links, I let the algorithm crunch for a while. You are getting a maximum rate for each end to end requirement of 0.000534 packets/second. With 40*39 identical requirements (negelcting the W3IWI type choke points) this gives a network capacity of 0.833 packets per second or 9 million characters per day. This is of course awful. The most general network that can be analyzed by this technique is contained in the following

Theorem: The product form throughput evaluation and the computational procedure in Markovian CSMA networks with perfect capture and exponential packet length hold for arbitrary packet length distributions having rational Laplace transforms which need not be the same for all neighbors of a node. The analysis depends on the average normalized scheduling rate of nodes, independent of particular packet length distributions.

The bottom line is pretty clear. We have been overly generous to the network in that we neglected the real world of 0.08 bit/Hz modems run through VERY poorly equalized radios and we didn't even clobber the returning acks!! We MUST move to BTMA or some similar scheme before we grow any larger.

[1] Boorstyn, R.R., et.al, "Throughput Analysis in CSMA Packet Radio Networks", *IEEE Transactions on Communications*, Vol. COM-35, No.3, March 1987.

DSP Modems
It's Only Software

by Dr. Robert W. McGwier, N4HY

Are you as tired of buying new modems as I am? There is an alternative and I think a superior one in several regards. From the title, and the joint paper with Tom W3IWI[1], it should be obvious that we are talking about doing modems on DSP engines. Using a single chip processor as the basic building block, a low cost, high speed modem can be programmed to operate through *unconditioned* radios. If you want a different modem ten minutes later, you just change the software driving the chip. The primary advantages stated more explicitly are (1) a single piece of hardware can make and demodulate the phase or frequency modulated synchronous signal of your choice, and (2) the software can do very effective adaptive equalization to ameliorate the bad things the radio (filtering) is doing to your data signal.

Lets take an example and go through a description of what software we need on the Texas Instruments TMS320 to do the modem functions for a. a FO-12 BiPhase PSK demodulator. We wish to use adaptive equalization and a fairly general demodulation scheme that can apply to other types of phase modulation schemes as well (QPSK, etc.). We will work our way "backwards" in the demodulation process from carrier generation to phase detection of a PSK signal. This is in order of increasing mathematical complexity but not necessarily computational complexity.

Sine Wave or Tone Generator

Common to both the receiver and transmit side of a modem on this chip is a sine wave generator. The process behind generating a sine wave or tone is fairly straightforward. It is usually called frequency synthesis. We store a table of values of sine at 64 values from the first quadrant. Stored will be the values of sin(X), where X = $0, \pi/128, 2*\pi/128, \ldots, 63*pi/128$ radians. For finer resolution, we can store a table of values $sin(Y)$, where Y = $0, \pi/8192, 2*\pi/8192, \ldots, 63*\pi/8192$. between 0 and n/128 or do interpolation. We will use the second table approach and do mod 16384 arithmetic for calculating angles. The highest order two bits are quadrant, the remaining twelve bits will be used to index into the the gross and fine value table as will be described below. For angles outside the first quadrant, or cosine, or to use the fine table approach we use a fundamental trigonometric identity

$$(1) \quad sin(X+Y) = sin(X)cos(Y) + sin(Y)cos(X).$$

In our dual table approach at frequency sysnthesis, let X one of the gross values indexed by the six bits just below the quadrant bits of the current phase. Y will be a fine resolution angle got ten from the low order six bits of the phase. Since Y is small, a moment on your calculator will show you that $|cos(Y)-1.0$ is so close to zero that we will replace $cos(Y)$ with 1 in our formula. Y is always between 0 and $\pi/128$. Our formula' becomes

$$(2) \quad sin(X+Y) = sin(X) + sin(Y)cos(X).$$

where sin(Y) is read out of the fine table and $sin(X)$ and $cos(X)$ are read from the coarse table. Let Φ be the phase of a sine wave or carrier. To get a tone of constant frequency, we just add a constant value at each sample time to the old value of Φ. We wrap back to zero when the Φ value goes past 2π which is 16384 in our arithmetic. The size of the value we add between each sample output value corresponds directly to the frequency since it meaures how fast we wrap ourselves around the circle. The implementation we have on the chip now gives fourteen bit resolution of the circle and 0.5 Hz frequency resolution from 0.5Hz to about 4 Khz given a sampling rate of 8.192 Khz. We can use this algorithm to generate FSK, M-ary PSK, and others.

Carrier Tracking

To demodulate a signal carrying digital data by means of a phase shift keying approach, we need to know the carrier frequency and phase so that we can do noise rejection by using narrow filters and to recover the clocking of the digital data so that we can reasonably expect to be able to distinguish a zero from a one. In the PSK modem used to demodulate JAS-1/FO-12 both the Clark/TAPR adaption of the JAMSAT modem[2] and James Miller's PSK modem[3] use a *phase locked* loop, hereinafter, PLL. Gardner[4] has written the definitive treatise on analog PLL's. In it can be seen how much effort has gone into making analog PLL's work. When done in software on these DSP engines, many of the difficulties inherent in making an analog loop disappear. In a digital representation, you can make an attempt at doing a minimum variance unbiased estimate of the current frequency and phase rather than trying to zero some phase error. The free running frequency of a digital "VCO" is wherever the frequency was last estimated to be located in a second order PLL (the only type we will consider). So there is not this tendency to pull away and back to the "natural" free running frequency of the VCO in an analog implementation. Thus temporary loss of lock is not as disastrous as in the analog implement ations.

Our modem receiver which will be a PLL is a trivial program on DSP chips. The tone generator we described earlier will be our VCO. Suppose our incoming complex signal is S_r and the VCO produces S_0. Our "phase de-

tector" is simply $S_r - S_0$ = AS. Ask W3IWI how much effort he expended in getting just the right parts and setup so that he would have no DC offset in the phase detector in JAMSAT/TAPR PSK modem. We feed this through a loop filter, which when combined with natural integrator in our tone generator will be our second order loop filter. Take our current estimate of the frequency, the constant we add to the-phase at each sample time, and add to it a constant k_1 * AS and the old phase. This is the phase we will use at the next sample time. To make a new estimate of the current frequency we multiply the old estimate by k_2 and add it to k_3 *AS. These values of k_i, $i = 1,2,3$ are (in Kalman filter parlance) called "gains." Simply put, it is how much we wish to believe the current phase error AS when making changes to our estimates of the phase and frequency. The updated frequency estimate just determined is fed to the tone generator for the next sample period and the process is repeated. This phase locked loop is 13 instructions on the T.I. TMS32010, and with the implementation we are using (Delanco-Spry Model 10) the processor is running at 6.25 Mhz. The phase detector and loop filter takes 2 μs. When this is combined with the sine generator algorithm, the total time for the entire PLL algorithm per sample is 17 μs. A block diagram of the PLL algorithm from the phase detector to the VCO output is given in figure 1.

Demodulation

It may seem odd that in attempting to extract zero's and one's from a PSK modulated carrier, that we must somehow remove the effect of the modulation in some way to be able to make adequate decisions about which bit is being transmit ted. Let S,(t) represent the received signal at time t. To do this, this the extracted carrier is split into in-phase (I) and quadrature (Q) components by use of a Hilbert transform. We will call this S,(t) and $\hat{S}_r(t)$ respectively. A discrete-time Hilbert transform is a way of taking a sequence of comples signal samples and rotating the phase of the Fourier components forward by 90^o. This is very difficult to do over a range of frequencies in analog devices but is easily implemented in software by using an FIR (finite impulse response) filter[5]. Simply put, an FIR filter is a sequence of N numbers that you multiply the by the preceding N signal samples at each time sample t to get a filtered output.

$$y(t) = \sum_{i=0}^{N} x(N - i) * h(i) \qquad (3)$$

where $y(t)$ is the filter output, $x(N$ -i) are the current and previous N signal samples and the h's are the filter coefficients. This formula is "THE" formula is DSP software. This structure is used to produce bandpass filters (low, high, complex bandpass, etc.) and the adaptive equalization filters whose coefficients *h axe* allowed to change in time according to some preset rules. It is the same formula used to compute discrete Fourier transforms. The only place we do not use FIR filters in this modem will be in the loop filter described above. The schematic found in [5] for these filters is in figure 2. The coefficients are for a bandpass filter and are found using the Remez Exchange Algorithm[5]. Rather than go through the mathematical formulae associated with the Hilbert transform demodulator, we give it to you in block diagram form in figure 3.

Referring to figure& the (Q) channel output, $\hat{S}_r(t)$ is mixed with carrier references I and Q and the (I) channel (the unshifted input samples). The real and imaginary parts of the signal are taken from the adaptive equalizer and "arctangent" is used to compute the phase angle. On the TMS32010, the arctangent is a table of 25 to 30 words along with a linear interpolation algorithm that will give the arctangent to a few tenths of a degree accuracy. The equalizer filter we will use will be a twenty tap (twenty coefficients) affair and its evaluation takes 20 μs. The outputs of the demodulator are phase angles which are used to feed the PLL. In figure 3, this demodulator output will replace the phase detector. Other outputs are from the real and imaginary leg equalizers are the data encodings and these are fed to a decision algorithm which decides which bit pattern we are trying to encode. This decision algorithm also output s the error from the nominal value of the encoding and this is used to modify the values in the adaptive equalizers. The arctangent evaluation takes 18 μs. Most modems use scramblers. This does NOT mean encrypt ion! This means we are trying to prevent bad sequences from occuring and causing the demodulator to lose the ability to follow the baud transitions (clock). We can implement a seven stage shift register in 18μs and the bit error rate will be improved considerably.

We are just beginning the modem development work and if you have some expertise in the area, please let us know so that we can move more quickly toward a nice product for the amateur community.

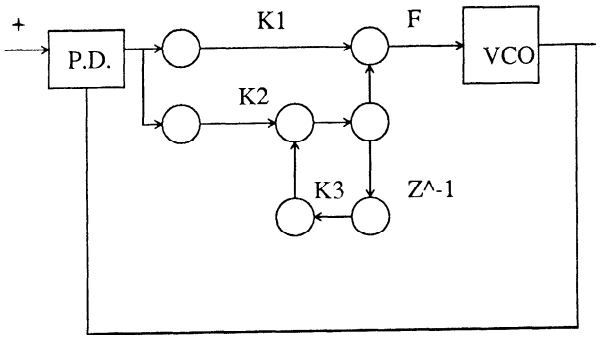
[1] Clark and McGwier, "Digit al Signal Processing", this volume.

[2] JAMSAT, "A PSK Modem for JAS-l", QEX, April 1986.

[3] Available from AMSAT-UK.

[4] Gardner, "Phase Locked Techniques", Prentice Hall, 1972.

[5] Rabiner and Gold, "Digital Signal Processing", Prentice Hall.



Digital Phase Locked Loop

FIR Filter

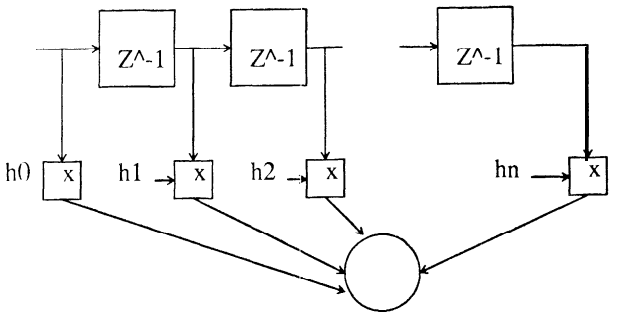


Figure 2

Hilbert Transform Demodulator for BPSK (M-PSK is done by doing modulo pi/m)

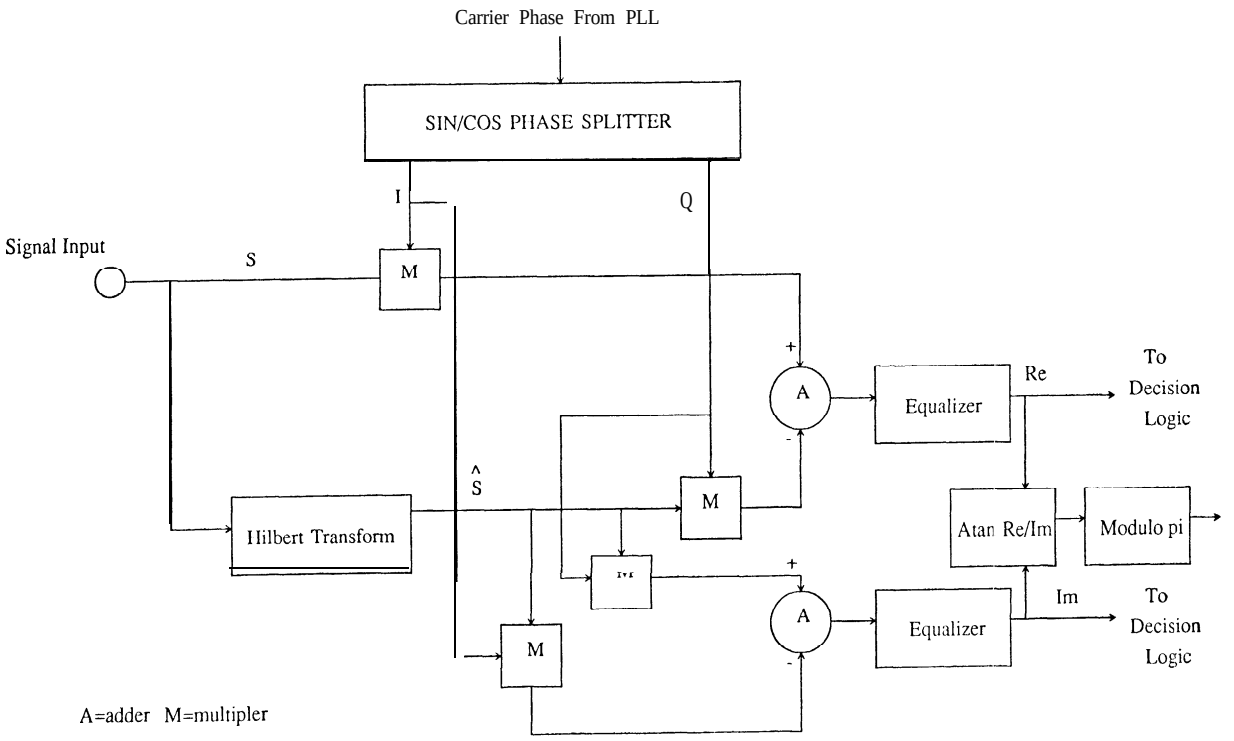


Figure 3

HF PACKET: WHERE DO WE GO FROM HERE?

Barry D. McLarnon, VE3JF
2696 Regina Street
Ottawa, Ont., Canada K2B 6Y1

Introduction

Packet radio on the HF bands is alive and well, and is steadily gaining in popularity. The HF links which have been established between widespread packet bulletin board systems have become a workhorse in moving error-free traffic beyond the limits of the VHF/UHF packet networks. These links will, of course, never be capable of handling huge volumes of traffic, like megabyte files: the bandwidth simply isn't sufficient. Satellites and expanded UHF/microwave links must be developed to meet these requirements; but it is probably safe to say that HF will always have a role to play in amateur data communications, both as a back-up to these higher-capacity (but more vulnerable to failure) systems, and for extending the network into remote areas where setting up a satellite station may not be feasible.

There is little doubt that HF packet will play an important role in amateur data communications for many years to come. On the other hand, even its most enthusiastic devotees would likely admit that the performance of the present AX.25 HF packet links is often disappointing. At times, they sail along so smoothly that they are reminiscent of VHF links (albeit at a lower data rate). At other times, for reasons which are often unclear to the users, the links bog down with retries or fail completely, in spite of what appears to be adequate propagation to support communications.

The reasons for this erratic performance can be broken down into three main areas:

(1) The unsuitability, in some respects, of the AX.25 protocol itself for the HF environment.

(2) Difficulties in applying the networking concept of multiple-access (channel-sharing) to the HF environment.

(3) Problems with the modulation schemes and reception techniques used to transmit the AX.25 frames over the HF channel.

In contrast to conventional RTTY and AMTOR, AX.25 is based upon a standard (CCITT Recommendation X.25) which was specifically designed for computer-to-computer communications. On the other hand, it was certainly not designed for use on the HF channel; its usual domain is the relatively benign environment of the Public Switched Data Network.

An important aspect of protocol performance is throughput efficiency. AX.25 does not fare too well in this area, due to the large amount of overhead bits (bits other than information bits) contained in every packet. The overhead amounts to 152 bits, of which 112 are callsign information, assuming a point-to-point link without digipeaters. This is not a serious penalty when maximum-length packets (256*8 = 2048 information bits) are transmitted. Unfortunately, on HF channels the probability of receiving a packet without errors tends to fall off rapidly with increasing packet size, and in practice, much shorter packets must normally be used. The overhead then becomes an appreciable fraction of the total packet length, and the throughput suffers accordingly. Another problem with AX.25 is its inability to take good advantage of longer, multiple-frame transmissions, which reduce the overhead due to turnaround time (transmit/receive switching and transmission of ACK packets). The limitation is a result of the lack of a selective repeat capability in the protocol. The structure of the AX.25 protocol also does not lend itself to the use of signal processing techniques (memory ARQ, forward error correction, soft-decision decoding) which allow error-free packets to be built up from several corrupted packets.

These and other aspects of data link protocol design for HF are treated in a longer version of this paper which has been submitted to the ARRL for publication. Space does permit including the discussion here; nor, I suspect, would the prospect of creating Yet Another Packet Protocol (apologies to WA7MBL) be greeted with widespread enthusiasm in the packet community! The remainder of this paper will be confined to discussing performance improvements which are applicable to AX.25 HF links.

The next reason for poor performance of HF

packet links is more a function of usage than protocol design. Packet allows the sharing of channels, by virtue of its CSMA (carrier-sense multiple access) capability. This leads to a tendency for users to congregate on a small number of channels, which is not in itself a bad thing; for low-density traffic like keyboard-to--keyboard chitchat, it can make more efficient use of the limited HF spectrum available. Even for transmission of larger amounts of data, such as file transfers, the lower throughput caused by channel sharing may sometimes be acceptable. Unfortunately, rather than degrading gracefully, the throughput tends to rapidly fall to zero as the number of users on the channel increases. The reason for this is that the collision avoidance mechanism is imperfect. Collisions can occur for many reasons; the colliding packet that zaps yours may come from a "hidden" station in your skip zone, or because a fade caused the carrier detect to fail momentarily. Carrier detect circuits need a certain amount of time to respond; you and the other station may have both started to transmit during that response time "window" (random backoff for retries does help prevent repeated collisions of this type). Possibly the other station isn't detecting you because its receiver is mistuned. Whatever the reason, collisions in multiple-access channels are a major impediment to getting any sort of reasonable throughput.

It also should be mentioned at this point that it is often unclear, particularly to newcomers to the mode, how to set their TNC parameters for best performance on HF. Some rules of thumb have emerged, such as keeping packets short (80 characters or less) and sending only one or two frames per transmission, but the optimum parameters will vary widely with conditions. There is considerable latitude here for experienced operators to "fine tune" their parameters as conditions change. The best bet for newcomers is to check with some of the HF packet gurus and find out what has worked best for them. Some aspects of HF propagation, such as the MUF for a given path, are fairly predictable. An interesting open question for HF BBS operators in particular is the extent to which optimum TNC parameters can be predicted and included in their forwarding files, or perhaps even adapted dynamically.

The balance of this paper will deal with the third topic mentioned above, the modems and associated RF systems used in HF AX.25 systems. First we need a bit of background on HF channel characteristics, and how they affect data communications.

The HF Channel and Modem Design

A large part of the problem with HF packet rests with the design of the modem, which was adapted from the Bell 103 standard. Stated simply, this modem is not capable of reliable communications at 300 bps under the variety of conditions encountered on HF channels. Signalling using binary FSK at this rate produces a symbol length (bit duration) of 3.3 ms. Unfortunately, the signal received at the far end of the link does not usually arrive by means of a single ionospheric mode; instead, it is a superposition of several replicas of the transmitted signal which have travelled by different routes (e.g., they may have undergone different numbers of hops and/or reflected from different layers), and consequently arrive at slightly different times. This phenomenon, known as multipath propagation, is virtually always present to some degree; it results in noticeable distortion to voice signals, but its effects can be much more catastrophic for data signals. It leads to a form of self-interference called intersymbol interference, in which a bit can be demodulated in error due to the delayed energy arriving from the previous bit(s). QRM is bad enough on the bands without doing it to yourself:!

The degree of multipath present in a received signal is measured by a parameter called the multipath spread. Its exact definition need not concern us here; the important thing to know is that when the multipath spread increases beyond a few per cent of the symbol time, the bit error rate performance of the FSK system begins to deteriorate rapidly. When the multipath spread exceeds about 10% of symbol time, it becomes the dominant mechanism in controlling the bit error rate. In other words, if you are operating in this region, improving the signal-to-noise ratio at the receiver (by increasing transmitter power or antenna gain, for example) will produce no significant improvement in error rate! Obviously one should attempt to avoid this situation as much as possible.

Measured data on multipath spreads are not easy to come by, but the author has made a number of observations on short-range paths (60 - 1000 km) in connection with HF data system tests, and would estimate the average spread to be in the neighborhood of 1 ms, with values up to 3 or 4 ms not uncommon. Some data on longer-haul paths have been published (Ref. 1). For example, measurements taken over a four-year period on the 6000 km path between Washington, D.C. and London show an average multipath spread of about 1.3 ms. The observed spread was less than 1 ms for only 30% of the time, and it exceeded 3 ms for 5% of the total period. Similar observations on the 9600 km path

between Tokyo and London yielded even higher values: an average spread of about 2.4 ms, less than 1 ms for only 5% of the time, and greater than 3 ms for a whopping 19% of the time!

In addition, there are other impairments which increase in severity with decreasing symbol time. One example is Doppler spreading, a phenomenon most prevalent on signals which traverse the aurora1 zones; it results in the well-known "Arctic flutter" effect. Interference from other stations can also be expected to be more severe as one increases the bandwidth in order to accommodate faster signalling rates. The conclusion is clear: whether the path is short or long, a conventional FSK modem running at 300 bps is not going to deliver a useable error rate for a significant proportion of the time that the band is open and providing an adequate signal level. During these times, increasing transmitter power or antenna gain will NOT help. It is nice to be able to run at 300 bps when the channel supports it, but one should be prepared to fall back to a lower rate, say 75 or 100 bps, when it does not. Better yet, the 300 bps modem should be designed to perform just as well as the lower-speed modem!

The nastiness of the HF channel leads one to ask the question, are all nonamateur HF data systems restricted to very low data rates? In fact, the vast majority of them do operate at rates in the neighborhood of 100 bps. To go higher, one not only pays a price in bandwidth, but the price of a suitable modem tends to rise exponentially with bit rate. HF modems are certainly available for rates up to at least 2400 bps, but the main customers are military, to whom cost is seldom an overriding concern! However, the increasing availability and decreasing cost of digital signal processing components should help to bring the techniques involved within reach of the amateur fraternity. The problem of limited bandwidth available in the amateur bands is a much more serious limitation, as the higher data rate signals would occupy the equivalent of a voice channel. The relatively wideband emissions associated with the 1200 bps and higher-speed modems would not be welcome additions to the congested HF bands, and should not really be necessary in the long run as high-speed satellite and terrestrial links and the higher levels of networking to support them become available. At the present time, it might be wiser to focus on the design of a high-performance 300 bps modem than to charge off in a quest for higher data rates.

Parallel Modems

Commercial HF modems operating at medium and high bit rates (i.e., 300 bps and up) can be roughly categorized as parallel or serial. Parallel modems simply multiplex the data stream into several low-speed subchannels which are spaced just far enough apart in frequency to avoid interfering with each other. A prominent (and very costly) example is the modem defined in the military specification MIL-STD-188C, which uses a total of 16 subchannels spaced 170 Hz apart, plus an additional tone used for correction of tuning errors and Doppler shift. Each subchannel has a basic symbol length of 13.3 ms, which normally would correspond to a rate of 75 bps per subchannel; however, the modulation is four-phase PSK, which allows two bits per symbol to be transmitted, for a total data rate of 2*75*16 = 2400 bps. In practice, this modem is often used in an "inband frequency diversity" mode in which the same data bit is transmitted on two or more subchannels simultaneously. This ploy lowers the effective data rate but increases the probability of demodulating the bit correctly (see discussion of diversity reception below).

Application of the parallel modem concept to a 300 bps design would be straightforward. For example, consider the 100 bps rate recommended above for a single FSK channel; for this rate, the recommendations of the CCIR (Ref. 2) for frequency shift and spacing between adjacent channels are 80 to 85 Hz, and 170 Hz, respectively. Thus we might have three parallel 100 bps FSK subchannels with center frequencies of 425, 595, and 765 Hz (these happen to be the first three recommended center frequencies, but many other choices are possible). The three subchannels could be easily constructed from separate FSK "building blocks" similar in design to those used presently by amateurs. The inputs to the three subchannels would be derived from a 1-out-of-3 data multiplexer, and the three outputs summed before being applied to the transmitter. An attractive alternative would be to implement all three subchannels with a single digital signal processor (DSP) chip. A modem using one of these devices would have the considerable advantage of needing no tuning whatsoever - it comes to life with all filters and oscillators perfectly tuned and stays that way! (some analog filtering is needed for anti-aliasing and reconstruction in the analog-to-digital-to-analog conversion processes, but this is relatively noncritical and should never require adjustment). The cost of DSP devices and their development tools has kept them out of amateur applications, but it just a matter of time before they will begin to make their presence felt; they are the wave of the future in low-frequency signal processing.

Serial Modems

The second major category of modem is the serial modem, which simply means that only one signal (normally a sinusoid modulated in frequency or phase) at a time is transmitted. Most telephone-type modems, and all modems presently used by Amateurs, fall into this category. Each signal (symbol) may represent a single bit of information, in which case the modulation technique is called binary (as in the 300 bps and 1200 bps binary FSK modems now used for AX.25 data links); in this case, the signal has two possible states, commonly called mark and space. Most 1200 bps and higher-rate modems produce signals with more than two states and thus carry more than one bit of information per symbol; otherwise, their spectra would not fit within a standard voice channel. Serial modems tend to have much shorter symbol lengths than parallel modems operating at the same bit rate, and therefore they require a more well-conditioned channel in order to avoid intersymbol interference. Most telephone-line modems include an equalizer in order to condition the channel;.it consists of a filter in front of the modem which is designed to flatten the amplitude and time delay response of the channel and thereby reduce the intersymbol interference and other distortion that result in reduced noise margins in the demodulator. Some modems, such as the very common 212A 1200 bps type, used a fixed equalizer design based upon typical telephone channel characteristics. More sophisticated higher-speed modems use an adaptive strategy: at the beginning of the call, a special "training sequence" is transmitted from each end of the circuit which enables the receiving modem to adjust the parameters of its equalizer for minimum distortion of the received data signal.

The principle of adaptive equalization also applies to HF modems, but successful implementation is much more difficult. Since the response of the channel is now time-varying, the equalizer parameters must be frequently updated. This generally means that the training sequence must be periodically reinserted into the data stream to allow re-adaptation. Modems which employ the training sequence technique are generally known as "reference-directed" adaptive modems. Other adaptation algorithms have been developed which do not require special sequences to be transmitted; for example, the equalizer can be adjusted to maximize the demodulator "eye pattern" opening without knowledge of the actual data sequence transmitted. This mode of operation is known as "decision-directed", and is of course more desirable due to the lack of overhead involved. Quite a number of attempts, some dating back to the mid-sixties, have been made to implement adaptive serial HF modems, mainly for the 2400 bps data rate. These designs share a common characteristic: when the channel is reasonably well-behaved (e.g., slow fading), they tend to perform well, sometimes even spectacularly; when the channel gets nasty, however, there always comes a point when the rate of adaptation is not sufficient to keep up with the fluctuations in the channel, and the modem fails equally spectacularly. In the latter case, a parallel modem may still deliver a usable error rate and therefore work over a wider range of conditions. On the other hand, the serial modem may offer higher overall throughput by virtue of better performance during the majority of the time, when the channel is not varying too rapidly. One reason for this better performance is the following: the serial modem transmits a single sinusoid at a time, and thus produces a more or less "constant envelope" signal; contrast this with the parallel modem output, which is a summation of several sinusoids. The parallel modem signal therefore has a higher peak-to-average ratio, and consequently will yield an output signal with lower average power from a typical peak power-limited transmitter (note that the clipping and compression techniques often used to overcome a similar problem with voice signals will not be well tolerated by the data signal!). All things being equal, the serial modem will then have more "sock" to cut through the noise and QRM when these are the primary limitations to communicating on the channel. Considerable effort continues to be expended on adaptive serial designs, and their performance should continue to increase as the state of the art in digital signal processing devices advances.

Although adaptive serial modems are presently difficult and costly to design and build, they should eventually find their way into amateur applications. Making one work well at 300 bps should be considerably less difficult than at 2400 bps and higher. (concentration on the higher data rates was stimulated in large part by a strong military interest in secure digitized speech). One intriguing possibility is the design of an adaptive equalizer to work with the presently-used 200 Hz shift 300 bps modems. Perhaps there is a well-heeled experimenter out there somewhere who needs a challenge!

Variable-rate Moderns

A useful concept which can be applied to both parallel and serial modems is that of the variable-rate modem. The basic idea here is that the channel capacity (the maximum rate at which information can be reliably transmitted) of an HF link with a given bandwidth is not fixed, but time-varying. In order to keep the link reliable, we should attempt to adjust our signalling rate to match the available

capacity. In contrast to a fixed-rate modem which is likely to collapse completely in the face of worsening conditions, the variable-rate modem allows the throughput to degrade gracefully. This concept is embodied in the Packet Adaptive Modem described by Rinaldo (Ref. 3). A much more sophisticated design, a parallel modem utilizing DSP techniques to provide six possible rates from 75 to 2400 bps, is described in Ref. 4.

Although simple in concept, the variable-rate modem is tricky to implement, the problem being to develop a suitable algorithm to monitor the system performance and carry out the necessary adaptation automatically. Performance monitoring is not too difficult, but any changes that ensue must be coordinated between the two ends of the link. This calls for a highly robust low speed link piggybacked onto the main data link; such a link is often termed an "order wire". Development of an effective variable-rate modem appears to be a worthwhile objective for the amateur community; in particular, a variable-rate serial adaptive modem would be less difficult to implement than a high-speed fixed-rate serial modem with adaptive equalizer, and it would avoid the peak-power limitations of the parallel modem.

HF Receiver Design

Some aspects of equipment design for packet operation, such as faster turnaround times, are beginning to be addressed by the manufacturers of amateur radio gear. Nevertheless, one suspects that they are less than fully cognizant of the requirements imposed by packet operation, particularly in the area of HF receiver design.

It is probably safe to say that the most important element of the HF receiver used for packet operation is the IF filter. This fact has been clearly demonstrated by Eric Gustafsen (Ref. 5) in his comparative study of HF modems. He also makes the important point that an audio filter, no matter how good, is not an adequate substitute for a suitably narrow IF filter. The crucial difference is that the audio filter will not prevent unwanted signals outside its passband from reaching the AGC detector, resulting in receiver desensitization and cross-modulation on the desired signal. The optimum IF filter bandwidth, of course, depends upon the type of data signal being received. For the 300 bps, 200 Hz shift binary FSK emission in present use, a study done in the early 70's (Ref. 6) indicates the optimum bandwidth should be about 360 Hz. This is a bit narrower than Eric's recommended range of 400 to 500 Hz. One good reason to use a wider than optimum filter at the present time is that available IF filters tend to have severe delay distortion (i.e., nonlinear phase characteristics), and this distortion is worst near the edges of the passband. This is a consequence of designing the filter for maximum possible rolloff rate in the stopband. The variation in delay over the passband of the filter can easily be several milliseconds, which can cause considerable intersymbol interference and consequent higher error rates in the data signal. This distortion could be reduced by means of a suitable equalizer, but hopefully this will become unnecessary as filters with characteristics more suited to data transmission become commonly available.

Another aspect of HF receiver design that is ripe for further study is optimization of the AGC system for packet transmissions. There is little doubt that the slow-release type of AGC time constant used for SSB reception is not very suitable for reception of data, especially when atmospheric noise is severe. However, it is not clear that the faster AGC characteristic commonly used for CW reception is that much better, or whether some other characteristic might be substantially better. In any case, the "optimum" is likely to be dependent on band conditions. Several speakers at an HF communications conference attended by the author in 1985 mentioned the dearth of knowledge concerning optimization of receiver AGC for data transmission. One stated that he had achieved better results by disabling the AGC entirely and carefully setting the RF gain manually.

Diversity Reception

Diversity reception is a technique which has found little application in amateur circles, in spite of the fact that its benefits have been known since the early days of radio. The reader is referred to the article by Nagle (Ref. 7) for a good overview of diversity techniques and their history; here we shall summarize them briefly and focus on their application to packet-type data communications.

Diversity reception might be defined as the processing of alternate versions of the same transmitted information in order to demodulate it more faithfully. The alternate versions may be generated at the transmit end by transmitting redundant information (in which case the technique bears more than a passing resemblance to error-correction coding!), or they may be generated solely at the receive end by sampling the received signal in two (or more) different ways. The first category includes frequency and time diversity, and the second includes space and polarization diversity. The key to success of the technique is that the different versions of the signal have encountered quite different perturbations during their

travels through the ionosphere, or, in mathematical parlance, that they are highly uncorrelated.

A straightforward and widely used (in commercial HF links) application of the diversity principle is frequency diversity, in which the same data is transmitted simultaneously on two or more separate frequencies. BY far the most common implementation is dual diversity, with two subcarriers carrying the same data (going to higher orders of duplication than two produces a state of rapidly diminishing returns). The separation required between the subcarrier signals to yield little or no correlation varys considerably with channel conditions; it may be tens of kiloHertz under extremely good, stable conditions, and as little as 100 Hz or so when conditions are very unstable. The minimum separation for decorrelated signal depends upon the multipath spread; a reasonably good "rule of thumb" holds that the separation should be at least one-half of the reciprocal of the multipath spread. For example, when the spread is 1 ms, the separation should be at least 500 Hz. In most cases, the separation used is of the order of 1 kHz; these implementations are known as "inband" frequency diversity, since the data subcarriers are contained within the bandwidth of a single voice channel. Such would probably be the case with any amateur implementations as well. Larger separations would provide better performance, but the technical and regulatory problems become more formidable as well. Even relatively small separations can give worthwhile performance gains, however. One study (Ref. 8) of inband frequency diversity yielded an average improvement in bit error rate of about one order of magnitude over single-channel operation. Such an improvement could result in a dramatic increase in system throughput. In this particular case, the data rate was 75 bps and the subcarrier frequency separation was 1360 Hz.

Returning for a moment to the parallel 300 bps modem proposed above, dual frequency diversity could be added in straightforward fashion by adding three more FSK subcarriers. The fourth subcarrier would carry the same data as the first, and so on. The simplest subcarrier frequency assignment would be to use the next three standard center frequencies, maintaining the 170 Hz spacing. This gives a separation between subcarriers carrying the same data of 510 Hz, which is considerably less than ideal; nevertheless, diversity gain would be available for a good deal of the time, and in particular when the multipath is severe. Other schemes are possible, such as spacing the two groups of subcarriers farther apart and creating a "hole" between them, which could then be occupied by other signals. However, special IF filtering would then be needed in the receiver to remove the unwanted signals in the "hole"; otherwise, these signals could reach the AGC detector and desensitize the receiver.

The next major category of diversity operation is time diversity. Here the same data is transmitted two or more times, with a time separation between the transmissions chosen such that the perturbations undergone by the signal are largely uncorrelated from one transmission to the next. Time separations of at least one or two seconds are generally required for this condition to hold. There are a number of practical problems in implementing a scheme of this type. In any case, it can be argued that a system which employs an ARQ protocol already has what amounts to time diversity built into it, and the interval between repetitions of a block of a data will almost certainly be sufficient to guarantee uncorrelated conditions. Furthkrmore, the ARQ system tends to adapt to the channel conditions, since the rate of repeats will be inversely related to the severity of the disturbances. When the channel is good, repeats will be few, and thus it will not suffer the penalty imposed by the fixed amount of redundancy in a simple time diversity scheme.

The next form of diversity reception we shall discuss, space diversity, has some intriguing possibilities. Space diversity involves the simultaneous reception and subsequent demodulation of the signal from two or more physically separated antennas. Once again, the aim is to derive uncorrelated versions of the signal, in this case by demodulating signals which have followed slightly different paths through the ionosphere and hence have undergone different perturbations. Here again, we have a technique which has been widely used in commercial HF applications for many years, and yet has been largely ignored by amateurs. Granted, the physical constraints imposed by many amateur installations may preclude the use of space diversity; nevertheless, the technique is within the grasp of many amateurs. Space diversity has one major advantage over frequency and time diversity: it does not involve the addition of redundant information to the transmitted signal. Since only the receiving set-up is changed, the technique could be applied immediately to existing data transmission techniques as well as future ones, and no regulatory hurdles need be overcome.

And now for the bad news (as usual, there's no free lunch!). In addition to two antennas (the use of dual diversity is assumed hereafter), you will need two receivers and two demodulators.

Duplication of the demodulator portion of the HF modem is not a major problem, but not everyone has two good-quality HF receivers in the shack. For those who do have the requisite receiving equipment, the next major consideration is the antennas. Other than being reasonably similar in gain properties, the primary requirement is that they be spaced far enough apart to yield a worthwhile diversity gain. How far apart is enough? Opinions vary, and actual measured data are scarce. Most textbooks state that the spacing should be nine or ten wavelengths; another (Ref. 9) states the minimum useful spacing to be four wavelengths. The 1985 ARRL Handbook, inexplicably, gives a value of only 3/8 of a wavelength as providing useful gain. The latter value is a bit hard to swallow, although Nagle (Ref. 7) does claim that good results have been obtained with a spacing of around one wavelength. In addition to reducing the potential diversity gain, however, very close spacing may cause problems with the matching of the antennas due to mutual coupling effects. The only thing that is certain is that you cannot have too much spacing, and should try for the maximum that is practical.

An interesting possibility to get around the constraints of small city lots and the need to own two sets of receiving equipment is to make an arrangement with a nearby buddy to use his station as a remote receiving site, and bringing the received audio back to your QTH via a telephone hook-up or low-power UHF link. The major stumbling block here is the need for some type of remote control of receiver tuning, but receivers with this capability are becoming increasingly common these days. Another solution to the antenna spacing problem of space diversity reception is to use a related technique, polarization diversity. A good deal of the fading experienced on the HF bands results from polarization mismatch between the receiving antenna and the signal, which is in turn caused by polarization rotation of the signal as it passes through the ionosphere. Combining the outputs of two co-located antennas, one having vertical polarization, and the other horizontal, can produce a marked decrease in fading and consequent lower error rate.

Having derived a pair of diversity signals by some means, it remains to combine them to produce a single output data stream. To begin with, each signal should be separately demodulated up to, but not including, the point at which a hard decision is made as to which data symbol was transmitted. The corresponding signal is that which is often observed in modem testing as an "eye pattern" (so-called due to the appearance of the signal when observed with an oscilloscope whose timebase is synchronized to the symbol timing of the received data signal). The "eye" signals from the separate diversity paths can be combined by means of one of three basic techniques: linear, selection, or maximal-ratio combining.

The three combining techniques have their theoretical pros and cons, but performance on the HF channel does not always subscribe to the theories! Linear combining is the easiest to implement, as the diversity signals are simply added together (typically with an op amp summing circuit) before being presented to the comparator or whatever circuit produces the binary output data. This simple scheme can work surprisingly well, but is not recommended for situations in which a diversity channel tends to produce a high noise output when the signal fades in that channel. This situation prevails, for example, in FSK demodulators using hard limiters, or in separate-receiver systems in which the receivers each have an independent AGC.

The next step up in complexity is selection combining, in which only one diversity channel is connected to the output decision circuit at any given time. An attempt is made to continuously monitor the signal strength in each diversity path and to rapidly switch to whichever is strongest. In practice, some hysteresis is built into the selection circuit in order to prevent excessive "hunting" back and forth between channels. Selection is clearly suboptimal in that potentially useful contributions to the decision process from the unused channel(s) are thrown away.

The third technique, maximal-ratio combining, in a sense combines the best features of the first two. The diversity signals are summed as in linear combining, but before summation the amplitudes of the signals are adjusted by multiplying them by a weighting factor which is proportional to the signal power in the corresponding channel. This approach makes the best possible use of all of the received signal information, but its theoretical advantages may not always materialize on real-world channels, and the complexity of the circuitry is considerable compared to the other methods. Nevertheless, the design of a maximal-ratio combiner is quite straightforward, and experience has shown that it will generally outperform the other methods by a small margin on the HF bands.

There has been a recent trend towards building more "intelligence" into diversity combining systems. A basic problem with most combiners is that the circuits which measure signal power to provide the basis for selection or maximal-ratio combining are usually "dumb"; that is, they cannot distinguish

the desired signal from noise and interference since they simply measure the total energy within a certain passband. This causes errors in selection or weighting to occur which can seriously degrade the performance of the diversity system. The key to overcoming this problem is to make the circuitry which assesses the diversity channels sensitive to certain known attributes of the desired signal. For example, a 100 bps "eye pattern" signal can be fed to a circuit which generates a fixed-length pulse for each zero-crossing of the signal. If the data signal is strong, the pulses will occur at 10 ms intervals, or integral multiples of 10 ms. The frequency spectrum of the pulse train will then tend to have its energy concentrated around 100 Hz and its harmonics. If the signal is dominated by noise and interference, on the other hand, the pulses will be more randomly distributed in time and the spectrum will not exhibit the same concentration of energy. A circuit consisting of two narrow bandpass filters centered on 100 Hz, followed by rectifers, lowpass smoothing filters, and a comparator can be used to distinguish between the two conditions. The output of such a circuit makes a more reliable signal quality assessor than a simple energy detector. Additional details on building intelligence into diversity systems are contained in Ref. 10.

Conclusions

The performance of HF packet systems can and should be improved. Considerable improvement in the performance of the present AX.25 system is possible through the design of better modems, improved HF receivers, and the use of diversity reception techniques. It is hoped that some of the techniques mentioned in this article will help point the way.

References

1. CCIR, Report 203-1, "Path time-delays and shifts caused by multipath propagation on radio circuits", CCIR Green Book, vol. 3, Geneva, 1982.

2. CCIR, Recommendation 436-2, "Arrangement of voice-frequency telegraph channels working at a modulation rate of about 100 bauds over HF radio circuits", CCIR Green Book, vol. 3, Geneva, 1982.

3. Paul Rinaldo, W4RI, "Introducing the Packet Adaptive Modem (PAM)", Second ARRL Amateur Radio Computer Networking Conference, March 19, 1983, pp. 2.71-2.74.

4. J.M. Perl, "Channel coding in a self-optimizing HF modem", Proceedings of the 1984 International Zurich Seminar on Digital Communications", Zurich, March 6-8, 1984, pp. 101-106.

5. Eric Gustafsen, N7CL, "'HF Modem Performance Comparisons", Packet Radio Magazine, vol. 2, nos. 1&2, pp. 21-24, January-February 1987.

6. R.T. Bobolin and J.C. Lindenlaub, "Distortion Analysis of Binary FSK", IEEE Trans. Communication Technology, vol. COM-19, pp. 478-486, August 1971.

7. John J. Nagle, K4KJ, "Diversity reception: an answer to high frequency signal fading", Ham Radio, November 1979, pp. 48-55.

8. P.McManamon and R.V. Janc, "An experimental comparison of nondiversity and dual frequency diversity for HF FSK modulation", IEEE Trans. Communication Technology, vol. COM-16, pp. 837-839, December 1968.

9. Keith Henney (ed.), Radio Engineering Handbook, 5th Edition. McGraw-Hill, 1959, p.19-118.

10. T. Gartell and D. Cawsey, "An intelligent diversity system for improved signal quality", Proceedings of the Conference on Communications Equipment and Systems, Birmingham, England, pp. 234-236, 1978.

FINDER - The Family INformation Database for Emergency Responders

A Multiple-Connect, Packet Radio Database for Emergency Communications

W. E. Moemer, WN61, Sharon Moerner, N6MWD, and David Palmer, N6KL
Santa Clara County Amateur Radio Emergency Service
c/o 1003 Belder Drive
San Jose, CA 95 120
408-997-3 195

Abstract

FINDER is a new packet radio database application utilizing multiple connect and a specific syntax to allow emergency response personnel to ascertain the status of their family members during a disaster. This paper describes the history, scope, operation, and design philosophy of the FINDER system.

1. Introduction

FINDER (the Family INformation Database for Emergency Responders) is a multiple-connect, packet radio database designed to allow emergency responders such as firefighters, policemen, or amateur radio operators to ascertain the status and whereabouts of their family members during a widespread disaster that overloads normal communications channels, thus allowing the emergency responders to more effectively perform their duties. Family members check-in with voice operators located at fire stations, who relay information to one of up to eight packet operators, all of whom are simultaneously connected to a central database station running the FINDER program. Emergency responders concerned about family members need only provide their home telephone number to a voice or packet operator to obtain a listing of the current information about all members of their family. FINDER can also be used in a "person tracking" mode to manage general information about victims of localized accidents such as train wrecks, plane crashes, or other multiple casualty incidents.

2. Purpose, Origins, and Scope

Part Five of the Amateur's Code reads: "The Amateur is Balanced...Radio is his hobby. He never allows it to interfere with any of the duties he owes to his home, his job, his school, or his community." Following a disaster, a ham's first concern is with his/her family, but the community also desperately needs his/her skills. FINDER is a way for the amateur radio operator to assist the community in such situations.

FINDER is an acronym for the Family INformation Database for Emergency Responders. It is a multiple-connect, packet radio database environment which enables emergency responders to determine the status of their families in the event of a widespread disaster. It is designed to address one problem that has long plagued emergency response organizations: in a widespread emergency that overloads the telephone system, how can the organization keep the emergency responders focused on the job when they may be distracted by concern for the welfare of their own family? FINDER offers a partial solution to this problem by using the capabilities of amateur radio and digital communications. FINDER allows emergency responders to ascertain their family's welfare without leaving their duties, enabling them to more effectively concentrate on the disaster at hand. This can all occur in the absence of commercial utilities and without tying up critical tactical communication channels.

2a. History of FINDER

Although city and county officials have always been concerned about the possibility of delay in the response time of trained personnel to a disaster, it was not until the Fall of **1984** that serious consideration was given by Santa Clara Valley ARES to this problem. Specifically, Bud Enochs, KE6DN, presented a written proposal for an Amateur Radio Emergency Health and Welfare Distributed Data Network in October of 1984. Despite this comprehensive document, the idea stayed relatively dormant until November of 1985 when Dick Rawson, N6CMJ, submitted a similar proposal for a program called the Emergency Worker Locator System. After a great deal of study, N6CMJ concluded that such a plan would not be feasible for Santa Clara County insofar as he calculated that each operator would have to be capable of handling one message every 25 seconds (this was based on staffing 80 firestations with **10,000** emergency workers with 2.7 family members each for a total of 37,000 messages being entered and **10,000** messages delivered).

Despite these pessimistic worst-case projections, subsequent experience in tracking riders in the annual Primavera Bicycle Tour proved that a multiple-connect packet radio database could successfully manage large amounts of real-time data. The Primavera database

program was written by Frank Kibbish, WB6MRQ, and handled the progress of approximately 2100 cyclists passing through five packet-equipped checkpoints [I].

The DEC at the time--Bill Robinson, WB6OML--had been keeping the county and city Emergency Managers apprised of the progress of the concept. Needless to say, the civil authorities were quite enthusiastic since it was clear that utilization of the hams could allow civil personnel to determine the status of their families during a disaster.

The project--now known as the Locator Project--picked up again in October of **1986** when WB6OML held a meeting for hams interested in devising a way to make the previous ideas work. At that time, Sharon Moerner, N6MWD, became the project manager and Dave Palmer, N6KL, and Weo Moerner, WN6I, took primary responsibility in writing the software that would make the idea a reality. Other hams became active on the committee (e.g., Don DeGroot, KA6TGE; Randy Miltier, N6HMO; Dick Rawson, N6CMJ; Glenn Thomas, WB6W; Don Tsusaki, WW6Z) and the project was ultimately renamed FINDER. The first demonstration of the software occurred in December, **1986,** and the first alpha test was held in January, 1987.

The first countywide test of FINDER occurred May 30, 1987. In a three hour time period, 24 of the county's 63 firestations were staffed by amateur radio operators; **154** people--hams and their families--checked into the system. Another county wide drill is scheduled as part of the Simulated Emergency Test on October 17, 1987 and it is planned to staff not only all the firestations but to also include the participation of city firefighters and their families.

2b. Current scope of FINDER

There has been considerable discussion concerning the type of traffic passed by FINDER since it could be viewed as either Health and Welfare traffic or as Priority traffic. The agencies hams serve determine the priority of the communications provided, and the information supplied by FINDER is considered to be of high priority by the Santa Clara County Fire Chiefs Association.

Obviously, there are clear possibilities for extending the coverage of FINDER, both geographically as well as demographically. At some time, FINDER may serve other civil and public agencies. Due to limitations of bandwidth and operator staffing, however, it is not feasible to allow general public access to FINDER. Even at this stage in the project, however, city and county officials are quite supportive and are working with the hams in training emergency responders and their families in utilization of the system.

In addition to FINDER's primary role following a large scale disaster, it may also prove useful during localized emergencies (e.g., plane crash, train wreck, toxic spill) which require tracking of victims and supplies. (For an example of the value of packet radio in disasters of this sort, see [2]) A special mode of operation of FINDER, called "health and welfare" mode, has been established especially for these situations. Activation of FINDER in a multiple casualty incident enables crucial information to be efficiently passed outside of the disaster zone to an area which has normal telephone service.

3. Overview of FINDER

The central element of the FINDER system is the FINDER software and database. Up to eight packet stations may be connected to the database simultaneously, and these packet stations serve as the input/output ports for entering information and status requests. Each packet operator also serves as the net control station for a group of voice operators located at fire stations. The voice operators interact directly with the emergency responders and their family members to receive current information and to deliver responses to their requests.

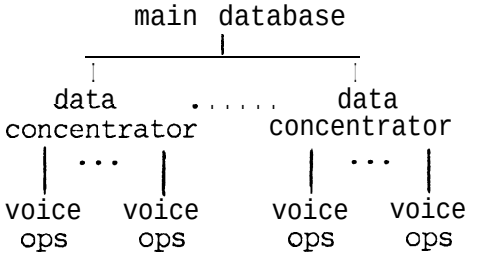
The FINDER software and database collects and organizes status information about emergency responders and their families. FINDER may be regarded as a specialized, multiple-connect BBS with a command set tailored to the handling of current information and status requests. Central to the concept is the fact that the various "concentrator" packet stations can connect to the database using any kind of AX.25 TNC and terminal that is convenient. In other words, all the specialized software (and hardware) that gives the FINDER system its personality is located at the main database station. This is in contrast to packet networks which require each user to have specialized hardware and/or or software in order to participate in the network. Such systems--while powerful--are not likely to be available in a large disaster such as an earthquake, because the number of operators with the required equipment may become vanishingly small.

FINDER works in a relatively simple way:

1. A family member goes to a nearby firehouse
2. He/she fills out the FINDER data card containing the home phone number, the first name, status (ranging from "All is OK" to "Contact as soon as possible"), and intended location
3. He/she turns the card in to the amateur radio operator on duty
4. The ham reads the information (via voice) to his/her designated packet operator
5. The packet op transmits the information to the database.

The information is retrieved when an emergency responder gives a voice operator his/her home phone number. The voice operator reads the status request to the packet operator. The packet operator keys in a simple search request to the database, and after a short delay, the database replies with the status of all persons sharing that home telephone number.

The following is a quick sketch of how amateur radio operators work with FINDER:



MAIN DATABASE: that packet station which serves to gather and store all the data which it receives from the data concentrators.

DATA CONCENTRATORS: packet operators simultaneously connected to the main database. Each concentrator will be typing in data received from his/her voice operators. Currently, there can be no more than eight data concentrators connected to the database at any one time.

VOICE OPS: voice operators located at fire stations throughout the county who utilize pre-designated simplex frequencies to communicate the information written on data cards to the data concentrator. The number of voice ops could range from two per data concentrator to as many as nine per data concentrator depending on the number of fire stations to be covered.

4. FINDER - The Program

The FINDER program is written in Turbo Pascal, with assembler code where needed for control of the serial port to the TNC. It is designed to run on an IBM Personal Computer@ or compatible, preferably with a hard disk, and requires a TNC (1 or 2) containing the firmware written by Ronald Raikes, WA8DED, which provides a "host mode" capability.

4a. Multiple-Connect

The multiple-connect feature is crucial to the efficient operation of FINDER. Multi-connect operation allows the available channel bandwidth to be used more efficiently, because at any given time many of the packet stations are involved with relatively slow keyboard input. This is in marked contrast to packet bulletin board systems (BBS's) in which only one station may be connected to the system at a time.

4b. WASDED Host Mode

WA8DED host mode is utilized in the FINDER program to greatly simplify the parsing of the data streams from the various connected channels. Basically, the TNC does not speak to the PC until it is told to do so by a polling command, and then the TNC does not send bursts longer than 256 characters in length. This removes the difficulties associated with asynchronous data streams and leaves the problem of queuing transmitted and received packets to the TNC itself. As a consequence, the FINDER program can concentrate on polling the connected channels, handling the sysop keyboard and display, and managing the database.

4c. The FINDER Database

FINDER uses the Turbo Database Toolbox ® [4] for all database entry, indexing, and maintenance functions. This useful package of Pascal functions and procedures implements a full B+ tree structure that generates a database file and associated index files on disk to aid in database searches [5]. FINDER implements two index files for the standard emergency responder mode of operation: a telephone number index and an origin index. This means that the database can be searched very quickly for all entries with a given phone number or with a given origination point. The database file can contain as many as 65,535 records, each 35 bytes in length.

4d. Svnopsis of FINDER command syntax

1. Single letter commands: The single character "h" followed by a carriage return implements the help function, and a brief reminder of the syntax is returned to the packet op. The single letter "b" (for bye) followed by a carriage return initiates a disconnect from the database machine.
2. General rules for data entry: The various fields are entered in order, with separators between them. Valid separators are the comma and the blank. Multiple consecutive separators are treated as a single separator. Case is ignored.
3. Syntax for CURRENT INFORMATION: (<cr> means carriage return)

   phonenum,aname,sl,orig,time,date<cr>

   See the FINDER DATA CARD (in the appendix) for an explanation of these fields. The first four fields are required, and the last two are optional.

   a. PHONENUM can have multiple embedded dashes; all dashes are ignored. Two general formats may be used: long form or short form. The long form uses the full ten digits for area code and phone number. The short form has only 8 digits, and uses a single leading digit as an abbreviation for the full area code. The allowed abbreviations are defined by the sysop in the configuration file. For example, "408" can be abbreviated "8", and "415" can be

abbreviated "5". The sysop may select no area code checking when the FINDER program is started. In this case, any ten digit phone number is accepted, but no 8-digit phone numbers.

b. ANAME is a 5 character field with a leading letter, signifying the first 5 letters of the person's first name. This identifies the specific family member.

c. SL is a 2-digit code, which is a combination of the "My current status is" and "I will be at" codes on the data card. The two digits should be entered one after the other, with no characters in between. Only the values shown on the data card are allowed.

d. ORIG is the 4-character code signifying the point of origin of the current information, i.e., the "This form filled out at" field on the data card. For instance, the code could be the fire station number where the person filled out the form. ORIG must start with a letter.

e. TIME is an optional field. If present, it must be a valid 24 hour time (with no colon). If this field is not present, the database program inserts the current time. TIME must be entered, however, if an entry for **DATE** follows.

f. DATE is an optional field. If present, it must be a valid day of the month. If this field is not present, the database program inserts the current day of the month.

g. Examples of data input:

85553195,joe,12,sj34<cr>
415-555-2368,mary,11,pa34,1235,3<cr>

These six pieces of information are stored in the database as a record of the status and location of the person at a particular time and date. Further CURRENT INFORMATION packets with the same PHONENUM and ANAME will supercede old information for that person.

4. Syntax for STATUS REQUEST:

/phonenum,voiceopid<cr>
or
?phonenum,voiceopid<cr>

This command line instructs the database machine to look in the database for ALL persons with the same phonenum. **A** status report, listing all six of the pieces of information in 3. above is sent back to the packet station. At the end of the report, the line "FINDER report for <voiceopid> done at <time>" is sent, which signifies that no more names match the requested phone number.

a. PHONENUM has the same restrictions as in 3.a. above.

b. VOICEOPID is a free-form field of any length up to 255 characters with no embedded blanks or commas. It serves as a reminder to the packet operator of which one of his/her voice ops transmitted the status request. It is not stored in the database. It may be the same as the origin codes in 3.d. above, or it may be the voiceop's callsign, or it may be any other convenient string of characters.

c. Examples of status request:

/8555-3195,w1aw<cr>
/408-5553456,w6xyz<cr>
/55558011,sj14<cr>

5. Search by origin code: In addition to the phone number search, FINDER also supports a search by origin code in the form

$orig,routereplyto

This produces a listing of all persons in the database whose current information originated at ORIG.

6. Users command: The users command in the form "users<cr>" returns a list of the callsigns of currently logged-on packet stations. The response is of the form:

At WN6I-1: N6KL, W6BB-3, AJ6T, WB6MRQ-7.

7. Tell command: The Tell command 3110ws connected packet stations to use FINDER as a conference bridge. For example:

Tell <callsign1 >[,<callsign2>][...] <message>

The <message> is sent to the callsign(s) specified. The special callsign "*" or "all" is used to send a message to all connected stations. The recipient stations receive the <message> prefaced with the time of day and the sending station's callsign, e.g.:

1630 N6KL> Has Resource Net moved to 146.94?

8. Message to sysop: Any other packet that does not start with a number, "/", "?", "tell." or "users" is treated as a message to the sysop. An acknowledgment message of the form
"Msg sent to sysop at <time>"
will be sent back to the packet operator. The sysop can also send messages to the packet operator.

4e. Special sysop commands

The sysop keyboard accepts current information input and search requests like any connected channel. The sysop can also send commands directly to the TNC by typing <ESC> and the command as usual. Some commands (such as "l" for list) can reference a specific channel. To set the channel for these commands, first type "<ESC>sn", where n is the channel number. For example, to force a disconnect on channel 3, the sysop

types "<ESC>s3 <cr> <ESC>d <cr>" The sysop can also send messages to connected channels by typing "<ESC>nmessage" where n is the channel number. If the printer is enabled by pressing a PF key, the printer echoes all sysop screen output.

Several useful database functions may also be performed by the sysop. A summary of the number of database entries for each ORIG or each CLOC can be produced by typing "s<cr>". A single record in the database can be listed if its record number is known by typing "1 nnn <cr > ", where nnn is the record number. All database entries can be listed by typing "1 all<cr>". Finally, a single record may be deleted by typing "d nnn<cr>", where nnn is the record number. For obvious reasons, all these commands are restricted to the sysop only.

5. An Important Extension - FINDER Health and Welfare Mode

In certain emergency situations, a record format more flexible than what has already been described is useful. In a multiple casualty incident, one might like to record a telephone number, full first and last names, a current location, plus perhaps a long message that might describe other important information. For these situations, a second operating mode of the FINDER system has been implemented, called the "health and welfare" mode, although the "person tracking" mode might be more descriptive. If FINDER were used for patient tracking, for example, the traffic would be regarded as priority traffic, so the utility of this mode extends beyond simple health and welfare.

To prevent confusion, the mode of FINDER described before this section will be called the "emergency responder" (ER) mode, while the extended mode will be called the "Health and Welfare" (HW) mode. The operation of the HW mode is fully analogous to the ER mode operation, except that the data record format is changed and several additional searches are allowed. The HW mode is selected by running a different batch file, "FINDHW", at startup, and the database, index, configuration, and journal files are distinct and separate from those for the ER mode.

5a. HW Mode Data Input Format

The HW mode data input format is well-suited to health and welfare traffic and patient/victim tracking. Because fewer records would be expected under these conditions, the various fields are generally longer and more flexible. The HW data input format looks like:

phonenum,lastname,firstname,cloc,message

where

1. PHONENUM is an abbreviated 8-digit or a full lo-digit phone number with syntax identical to that for the ER mode. It could be the victim's home phone number or the phone number of next-of-kin.
2. LASTNAME is the last name of the person, up to 10 characters long.
3. FIRSTNAME is the first name of the person, up to 10 characters long.
4. CLOC is a 4-character location code starting with a letter. It could be a hospital code or a shelter code defined in advance or in real time during the incident. This field is similar to the origin code field in the ER case, except that instead of stating the location where the FINDER data card was filled out, CLOC is more general.
5. MESSAGE is a free-form field up to 45 characters in length which may contain embedded blanks. It is intended for additional information about the person, such as name of nearest relative, injuries, etc. that is determined to be useful during the incident.
6. The current time and date are inserted automatically by the program.

5b. Searches Available in HW Mode

1 . /phonenum,routereplyto
   - produces a list of all entries with the same phonenum

2 . /lastname,routereplyto
   - searches by lastname, listing all with the same lastname

3 . $cloc,routereplyto
   - searches by cloc. As an example, this could give a listing of all persons sent to the same shelter or hospital

6. **Configuration file processing**

A number of run-time parameters must be selected by the sysop to start FINDER. These are collected in an ASCII file called FINDER.CFG (or FINDHW.CFG), which needs to be edited only once to specify such-parameters as the journaling/backup drive, journal file name, prompted or automatic program startup, the backup interval, the RS232 comm port, TNC baud rate, area code checking, and area code abbreviations.

7. Backups, Journals, and Data Recovery

Once a transaction has been saved on disk or diskette, a number of safeguards against data loss are taken automatically. For each backup or journal entry, the relevant files are opened, updated, and then immediately closed. First, a journal file is used to record each valid transaction at the time it is entered. Second, FINDER automatically makes periodic backup copies of the entire -database and the index files. This backup is made after every "n" transactions, where the value of "n" is specified by the sysop in the configuration file.

This backup also occurs following any disruption in the computer--TNC link, or at any time the sysop presses "F1" on the keyboard.

Recovery steps may be required in the event that the power is removed during a disk write operation. In this case, the main database file (or it's entry in the DOS directory) may become damaged, and some combination of the journal or backup database files may be needed to reconstruct the data. A utility program called "Rebuild" is supplied and provides a number of recovery options. Rebuild can append the contents of the journal to the previously saved version of the database, can rebuild the index files from a working database file, can construct a set of database and index files from a journal file, and can generate a printed copy of the database.

8. Computers and Packet Radio in Disaster Environments

Computers and packet radio networks can be somewhat fragile. They can be confusing to operate, they contain multiple sources of error (due to hardware, software, and users), and they do break. Because FINDER is designed to work under the worst possible circumstances, a number of precautions are built-in to insure that data will not be permanently lost, and that when failures do occur, the data can be reconstructed easily.

For each transaction entered by a connected packet station, the FINDER program responds with some kind of time-stamped acknowledgement [6]. Once a packet operator receives this acknowledgement, he/she knows that the transaction has been safely written to disk or diskette. The syntax of the commands described above is thoroughly checked so that many typing mistakes can be caught automatically. In case of entry errors, a brief message is sent that identifies the problem. The packet operator can check on the health of the FINDER program by sending a single carriage return <cr>. If FINDER is running correctly and actively polling for incoming data, it will respond with a single <cr>.

If the database is up and running, it is best for operators to transmit data as it arrives and avoid buffering large numbers of transactions in their terminals. This is because operators may miss any error messages that would be sent during the upload process if any of the transactions contain errors. In addition, if the packet link becomes severed in the middle of an upload, AX.25 does not make clear how many transactions may have been lost. (However, there is no penalty for re-entering duplicate data other than the time wasted).

Picking the correct set of TNC link parameters is an important art. For the same reasons that buffering is usually not helpful, it is best for user TNCs to send only one outstanding, un-acknowledged packet at a time (e.g. set MAXFRAME to 1). This keeps a packet station with a marginal path from saturating the channel with subsequent packets and retries that are likely to be discarded. Also, the nature of the user interaction in this application is such that MAXFRAME=1 is a natural setting [7].

The TNC's FRACK parameter controls the amount of time the TNC waits for a frame acknowledge. FRACK should be set high enough to guard against the "hidden transmitter" problem. In one drill a user's TNC continued to retry packets about once every second because it could not hear other stations. Since the database TNC heard most users directly, it sensed the channel was busy and was held off while his and other user's packets collided repeatedly. A FRACK value of 10 seconds or higher is probably about right. In any event, local implementers should determine a set of parameters that work best for their environment, and then encourage all packet stations to use the same set.

While the FINDER database can be configured to start automatically and run without intervention, the sysop plays an important role in keeping the system working well in the presence of marginal paths. The sysop display screen is updated continuously with the results of the TNC's link status for each channel. The display provides the number of:

- Receive frames not yet processed
- Frames sent to the TNC but not yet transmitted
- Frames transmitted but not yet acknowledged
- Retries on the current operation
- Link status messages not yet displayed
- The AX.25 link state.

By keeping close watch of the number of retries and the AX.25 link state, the sysop can spot which packet stations on the network may be suffering from marginal paths. In these cases, the sysop can recommend (either via voice or by packet message) that the packet operator make changes in his/her antennas, TNC parameters, or radios. The sysop can also force the logoff of any user and then allow reconnect via an alternate digipeater path. In the event that the (database computer has been down for a while and packet stations have been forced to buffer their data locally, the sysop may ask that stations take turns uploading data, especially in areas where the terrain would make the "hidden transmitter" phenomenon a problem.

When FINDER is activated, selecting which available packet stations will serve as packet concentrators is an important decision. The packet stations must have a good path to the database, and must also have good signals to their voice operators. As has been observed numerous times, if both the database packet frequency and the voice simplex frequencies are in the same amateur band, steps must be taken to minimize desense between the packet operator's

voice and data radios, For this reason, moving the database to, say, 220 MHz, is ideal if enough 220 MHz-equipped stations are available when the disaster strikes.

9. Getting Your Copy of FINDER

The FINDER program is in the public domain; permission is granted for non-profit, non-commercial use only. The FINDER program is available from WN6I or N6KL by sending a blank, formatted 5 1/4" or 3 1/2" diskette with SASE for return to you. If you wish to run FINDER without changing the code, the cost to you is the cost of the diskette and return postage. Note that a configuration file facility is included so that you can tailor certain parameters to your system without changing the code itself. If you want small changes in the code, attempts will be made to try to accommodate you. In any event, your comments and suggestions are welcome.

10. Conclusions

The FINDER system is ready to be utilized to enhance the response of emergency service personnel during a widespread disaster. The FINDER system is what it is today as a result of the many hours of work donated by the FINDER committee and by the hams who have participated in the operational tests. In addition, the support of the Santa Clara Valley ARES has been invaluable as has been the response shown by city and county officials familiar with FINDER.

---

1. Patty Winter, N6BIS, "Packet Radio in Emergency Communications," *QST*, Vol. 70, No. 9, Sept. 1986, p. 53.
2. Bob Bruninga, WB4APR, "Packet Radio at the Wreck of the Colonial," *Packet Radio Magazine*, Vol. 2, Nos. 1 and 2, Jan.-Feb. 1987, p. 13.
3. Paul T. Williamson, KB5MU, "WA8DED Host Mode User's Guide," 23 May 1986.
4. Turbo Pascal Database Toolbox, V. 1.2, Borland International, 1985.
5. The Turbo Database Toolbox does not provide user I/O interface functions such as screen and keyboard handling, so these functions had to be added to deal with the special nature of the packet radio environment.
6. This is in addition to the underlying AX.25 protocol acknowledgements that also occur, although typically we observe that the user acknowledgement message is immediately packaged as an AX.25 I-frame which--since it includes the send and receive sequence numbers--does not require that an additional Receive Ready ("ACK") packet be sent, thus the overhead for this additional safeguard is low.
7. See also the paper by Phil Kam, KA9Q, and Brian Lloyd, WB6RQN, "Link Layer Protocols Revisited," 5th ARRL Computer Networking Conference Proceedings, Orlando, Florida, March 9, 1986, p. 5-25.

FINDER Data Card

ARES FINDER - Family Information Database for Emergency Responders
(Complete one or both sides, as appropriate)

Fill out this side if requesting information about your family.

STATUS REQUEST

|_| 408
|_| 415 Home Phone: |_|_|_|-|_|_|_|_|

FOR RADIO OPERATOR USE ONLY:

Route reply to: |_|_|_|_|_|_|_|_|_|_|

Fill out this side if entering information about yourself.

CURRENT INFORMATION

|_| 408
|_| 415 Home Phone: |_|_|_|-|_|_|_|_|

First Name .....|_|_|_|-I_|

MY CURRENT STATUS (circle only one):

All is OK . ..*..... 1
OK--contact when convenient 2
Contact me as soon as possible . . 3

I WILL BE (circle only one):

At home 1
At work 2
At relative 3
At neighbor 4
At hospital 5
At school 6
At shelter 7
At previously agreed upon place.. 8
On assignment 9
Other 0

FOR RADIO OPERATOR USE ONLY:

This form filled out at: |_|_|_|_|
(Origination Point)

Time (optional): |_|_|_|_| (24 hour)

Today's Date (optional): |_|_|
(Day only)

(reverse side)

FINDER Data Card

ARES FINDER - Family Information Database for Emergency Responders

STATUS REPORT

Time Received	Telephone	Name	Status	Originate Point	Time	Date

Provided as a Public Service by Amateur Radio

DIAL "O" FOR OPERATOR
MESSAGE ROUTING IN THE AMATEUR PACKET NETWORK

Thomas A. Moulton, W2VY
The Radio Amateur Telecommunications Society
206 North Vivyen Street
Bergenfield, New Jersey 07621

Abstract

Over the past several years a variety of message routing systems have been used. The tendency has been to not step back from the problems and address fundamental issues. This paper will attempt to achieve that elusive goal and outline a routing system which will allow the Amateur community to operate with efficiency and ease.

Issue

The Amateur packet community has used the RLI-style addressing format of 'STATION @ SYSTEM" for several years, This was NOT the way it began...

The first format provided a simple "TO" field ("STATION"]. This format required every message system to maintain a routing table entry for ALL USERS of the network. This rapidly became unworkable, and evolved to a system where the message systems would only be required to maintain entries for ALL OTHER SYSTEMS in the network. This approach gave us our current "STATION @ SYSTEM" format. This format was challenged at the time of introduction, but there was only one system implementation (and by extension, one implementor) to make the changes requested. What was clearly needed was a method for implementing IMPLICIT routing system.

Solution

The KA2BQE and NN2Z of the Radio Amateur Telecommunications Society have taken the CBBS code written by WORLI, VE3GYQ and others and produced PRMBS. They have made a variety of changes which inter-operate with the the WA7MBL message system. One of the more important changes has been the addition of a new message header structure. One component of this new header is a change made to the "TO" and "FROM" fields to provide IMPLICIT ROUTING of messages. They have implemented this as a suffix to the "SYSTEM". It takes the format:

STATION @ SYSTEM/AX.121 ADDRESS

This has been outlined in the RATS AX.121 document.

Because it is unclear whether all other systems will make comparable changes in the near-term, we have developed a scheme for implementing a subset of this approach on existing systems. It takes the format:

DESTINATION @ ADDRESS

The "DESTINATION" field should may contain the STATION callslgn or the GROUP or APPLICATION name. Examples of this include KA2BQE, NTS, RATS or ALL.

The "ADDRESS" field will contain the telephone AREA CODE or the STATE CODE. The AREA CODE is prefered, because there is a FREE 24-hour directory service maintained by the telephone companies (OPERATOR) which can provide this information. This information is also available in every telephone directory in the country.

For those who are unable to access these resources, we suggest the use of the STATE CODE. This two character field will provide an alternative form which is popular with some members of the Amateur community.

Both of these forms may be suffixed with additonal information. This is convenient for applications that require finer granularity. The AREA CODE may be suffixed with the local telephone exchange or node number. The STATE CODE may be suffixed with the first three digits of the postal "ZIP CODE". These suffixes add little administrative overhead in the message systems because they have a wildcard facility ("*"). For example systems in Colorado need only maintain entries of "201*", "609*" and "NJ*" for ALL messages bound for New Jersey.

Summary

This paper has outlined an approach to implicitly route messages through the Amateur Packet Network. The full proposal is already implemented on 50 PRMBSs throughout the country. The capability to support the interim subset on ALL OTHERS is available NOW. It can be accomplished through the addition of a few routing table entries. This proposal includes support for the popular addressing conventions without creating an administrative burden for the system operators. Further work will be required to help the user community understand and use these capabilities. RATS welcomes comments on this proposal.

Packet Radio and IP for the Unix[1] Operating System

Clifford Neuman, N1 DMM

Department of Computer Science
U niversity of Washington
Seattle, WA 98195

ABSTRACT

Many services are currently available on the ARPA Internet that would be of interest to amateur packet radio users. The ARPA Internet connects universities and other organizations around the world that speak TCP/IP. One advantage of running TCP/IP on packet radio is the ability to access these services, and to interconnect with other systems that are part of the internet. This paper describes the implementation of AX.25 as a link layer protocol for the Unix operating system and the use of this system as an IP gateway between our local amateur packet radio network and our department's ethernet at the University of Washington, which in turn provides access to the entire Internet. The potential role of such a system for amateur packet radio is discussed, and a mechanism to allow users that don't have the resources to run TCP/IP themselves to access such services is described.

1. Introduction

I n the past year, we have started to see wider acceptance and use of layer three protocols in amateur packet radio. So far, most of this activity has been by people who are interested in advancing the state of amateur packet radio. Many are people who deal with computer networks outside of amateur radio, and would like to see similar facilities available within packet radio. Others have worked on mechanisms to solve some of the problems that amateur packet radio produced, and their solutions have made a significant difference in the way people use packet radio in the parts of the country where their solutions are being tested.

In order to get more use of layer three protocols by the users instead of the developers, there are two requirements that should be met. First, they need incentive. There should be something that they can do using layer three protocols that they can't do using connection[2] mode. One incentive is the ability to access some of the services available on more established networks such as the ARPA Internet Among these services are nameservice, fi le transfer, access to various databases, a more flexible system for electronic mail, and the ability to log into hosts on connected networks. These services can be made available in two ways. Servers can exist directly on amateur packet radio hosts, or they can exist on other networks with a gateway set up between the two networks. By connecting our local packet radio subnet to the internet, it is possible to access files on, and log into, computers at other internet sites (or at least, those where we have accounts).

Secondly, we have to lower the cost of entry. Most packet users do not have IBM PCs, or computers of equivalent or greater power. Many are simply using terminals connected to their TNC. This is probably one of the reasons that NET/ROM is so popular. With a packet station and no special hardware, one is able to connect to a NET/ROM node, connect to another node through the network, and come out the other end. If we are to generate as much interest

[1] UNIX is a trademark of AT&T

[2] By "connection" mode we mean the existing connection mechanism provided with TNCs when higher level protocols are not being used.

in layer three protocols such as TCP/IP, then we must make it easy for connection mode users to connect to, and use, a system that speaks IP. We can then point out the advantages they would have if their own system spoke IP directly. Among these advantages are the abilities to exchange mail and transfer fi les while simultaneously connected to one or more other systems.

2. System Overview

We decided one way to approach the above problems would be to get a machine that is on our department's ethernet onto packet radio. We had a MicroVax-I[3] available for our use. The advantage of using such a machine is that it already supports many of the network services that are desirable in the packet radio community. Among these are electronic mail, remote login, fi le transfer, and name service. Although not presently running on the machine, there are other applications available too, such as NNTP[4] which could be used as a bulletin distribution mechanism.

The existence of such a system gives users who have brought up TCP/IP something to connect to. The next step is to give people who aren't yet running TCP/IP something with which they can connect. To do this, we want to allow users to connect to our system in connection mode, and for them to be able to login to our system by this mechanism. The only other service we want to support in connection mode is mail. We would like to be able to exchange mail with PBBSs, which don't speak TCP/IP.

3. Role in a packet network

A machine such as the one I described above serves several functions in a packet radio network. It functions as a server for various network services. It is useful as a "home" machine for those users who do not have computers of their own. It also can serve as a gateway between multiple packet subnets, and perhaps even non-amateur networks. I have already described its use as a server. In this section I describe its use in some of these other functions.

3.1. Home machines

For those users who don't have IP running, our machine can serve as a home machine. Users can connect to it by using connection mode. The system can support multiple connections of this type simultaneously. When a user is connected to our system, he can use the various services available to IP hosts. He also will be allocated a limited amount of disk space, and will be able to retrieve fi les in which he is interested. The mail interface the user will be able to use presents a better interface than the central BBOARD mechanism which is currently in use. The user will be able to store messages indefinitely as long as he doesn't exceed his quota.

3.2. Level 2 to Level 3 Gateway

Since the user can connect to the system using connection mode, and since the system also speaks TCP, the system serves as a Level 2 to Level 3 gateway. Users will not have to give up connectivity with the old in order to begin using the new.

3.3. IP Gateway

A machine such as the one described above is also a logical machine to use as an IP gateway, at least until such time as we have dedicated machines for such purposes. Gatewaying could be between multiple packet radio networks, and even between non-radio networks such as the ARPA Internet

There are services available on the ARPA Internet that are of interest to packet radio users. If there is a university in the area, it is likely that there may be an online database of upcoming events. There are also many mailing lists on the Internet that might be of interest to Amateurs.

Connecting to non-amateur networks does bring up a number of issues, such as screening of messages in both directions. I discuss solutions to this problem later in this paper.

3.4. NET/ROM

NET/ROM fits in nicely with TCP/IP. IP can be run on top of NET/ROM. In such an arrangement, users on a LAN would speak IP on top of AX25. Multiple LANs could then be linked together using NET/ROM. An IP gateway would exist on each local area network and would appear as a NET/ROM node to other NET/ROM stations. This arrangement is similar to the way that local area networks are linked by the

[3] MicroVax is a trademark of Digital Equipment Corporation

[4] Netnews Transfer Protocol

ARPAnet. NET/ROM nodes correspond to the IMPs on net IO.

4. Related Work

There has been a lot of work recently in the TCP/IP arena. Work has been done on Phil Karn's IBM-PC code, and it has been ported to other machines such as the Amiga, the Mac, and others. Steve Ward and Mike Chepponis have been working on additional features in order to give users greater incentive to upgrade to TCP/IP.

Implementations of the TCP/IP code are needed for many more machines. Services such as the ones I have described also are needed for these machines. Not many people have access to a MicroVax as I did. It is a good machine to use in order to determine how network users react to such services. The more machines such services are available on, the more people will be able to set them up.

5. Implementation

The Ultrix[5] kernel already had all the code necessary for Internet Protocol. Because we did not modify the "upper" IP interface, layers riding on top of IP were able to use the packet radio medium without modification. Thus, TCP and UDP did not need to be modified and, similarly, applications running on top of those protocols worked without modification. The IP code in the kernel did not require modification either. All we had to do was to find a way to take the IP packets generated by the kernel, encapsulate them in AX.25 packets, and send them off, using SLIP, to the KISS interface of the TNC.

5.1. IP and AX.25 and the gateway

We chose to implement a pseudo-device driver for the packet radio interface. The driver supports the same calls as network device drivers do for other media such as ethernet Our driver is a pseudo driver because there is not really any hardware on the bus for our packet radio controller. Instead, our controller is plugged into a dz[6] port, and the kernel must communicate with it through that port

Teaching the kernel to recognize the new interface was easy. There is a structure called if-net that is associated with each interface. This structure contains pointers to the kernel procedures, which are used to initialize the interface, send a packet, change parameters, and a few other operations. The next trick was to figure out how we could receive packets. This was done by including a routine similar to the one that gets called in the ethernet driver when a packet arrives. The difference, though, is that our routine is called by the dz driver whenever a character is received on the line to which the TNC is connected.

As each character is read, we do some initial processing on the fly. In particular, we unescape frame end characters that are embedded in the packet. When the final frame end is read, we check the header of the message, note the callsigns, note the layer three protocol type, and if it is IP, we add the encapsulated IP packet to the queue of incoming IP packets to be dealt with by the existing upper layers.

In order to implement the routines described above, we started with a few routines from Phil Karn's code for the IBM PC. These routines encapsulated and decapsulated AX.25 packets. With a few modifications these routines were made to work in the Ultrix kernel.

The gateway functionality came for free. The way an IP gateway works is that when a packet is received, the system looks at its IP header to determine the destination address. If the destination address is not its own, it then decides which is the correct destination interface, and which system is the correct next hop. This is all done at the IP layer, and the same code that existed for gatewaying packets on ethernets works for AX.25 subnets too.

5.2. Address Resolution Protocol

The final task was to translate internet addresses into AX25 addresses. This is done using ARP, the address resolution protocol, in the same manner that IP addresses are translated into ethernet addresses. But, AX25 addresses look like amateur radio callsigns followed by a 4 bit system ID. To make matters worse, some entries may contain additional callsigns for digipeaters that are to repeat the packet Thus, what is needed is a different set of ARP routines for the packet radio interfaces. Phil Karn's IBM-PC code includes an ARP implementation that supports both AX25 and ethernet addresses. Because we did not want to modify the code for our system that is

[5] Ultrix is a trademark of Digital Equipment Corporation

[6] A controller for multiple RS-232 ports

used on the ethernet side, we decided not to take this code. ARP lookup occurs at layer two, and thus, gets called inside either the ethernet driver, or the AX.25 driver. The routing tables at the IP layer determine which driver is called. Since the ARP lookup occurs inside our code, we are able to call a separate routine that deals specifically with AX.25 addresses.

5.3. Connection mode

As already discussed, we would like to support connection mode on our gateway. Doing so would allow users who do not have the resources to run TCP/IP to be able to access IP network services. Further, users can give IP a try, and if they like it, then they might consider running it themselves. However, there is no reason, though, that connection mode should be supported in the kernel as is IP.

The way our implementation is set up, it is easy to allow user level process deal with connection mode. We can tell the kernel that if a packet comes in, and its protocol ID is not IP, that the packet should be placed on the input queue for the appropriate tty line. A user program can then read packets that the system isn't interested in from that line, and deal with the packets itself. By setting appropriate parameters for the kernel, additional filtering could be provided, though one would not want to do anything too complex in the kernel.

The user level process that reads such packets would have to keep track of any connections and support connection mode itself. Such a program could maintain multiple connections, and direct input to and output from pseudo terminals. This would allow connection mode users to log into the system. Such a program could accept connections to multiple SSIDs, thus allowing one SSID to be used for the transfer of mail with local non-IP bulletin boards.

5.4. Other layer 3 protocols

In addition to supporting connection mode, support could be provided in a similar manner for other layer 3 protocols. I already mentioned how NET/ROM can be used to forward IP packets. One could conceivably support the rest of the NET/ROM interface in the same manner as connection mode is supported. Of course, NET/ROM users would not have the benefit of the additional services available using IP.

6. Unresolved issues

The ability to interconnect amateur packet radio networks and non-amateur networks introduces a few problems which have not been completely resolved as of this time. In this section, I present those problems, and for some of them, I suggest some possible solutions.

6.1. Timeouts

One problem that comes up is the difference in bandwidth for the two networks. Hosts on the ethernet side expect fast response, and if they don't get a response quickly, they time out and retry their transmission. We have found that when connected to a system on our department's ethernet from a machine on the packet side of the gateway, the system on the ethernet side initially retransmits packets several times before a response makes it back. This results in wasted bandwidth on the radio side as the packet is needlessly retransmitted, and this in turn delays other packets. Fortunately for some implementations of TCP, once the connection has been established, the system on the ethernet side learns the correct timeout, and things settle down.

6.2. Internet routing

Routing is another problem that arises if we want to allow connections to internet hosts beyond our department's ethernet In order for a response to come back, all the gateways between the source and the destination must know the route to the appropriate packet radio subnet. Since a class 'A' network is allocated for AMPRnet, and since most systems by default will maintain a single route for a class 'A' network, only one path exists for all of AMPRnet, whereas what is desired is different: gateways for different subnets. It is conceivable that something like this could be handled using ICMP[7] redirects, but at this time, no mechanism is in place.

6.3. Access Control

Another problem we face is access control. Since operation is on frequencies assigned to the amateur radio service, any communication must be initiated by licensed amateurs. One way we can solve this is to maintain a table of authorized addresses on the non-amateur side of the gateway. Associated with each of these addresses is

[7] Internet Control Message Protocol

a list of hosts on the amateur side of the gateway with which that host can communicate. Initially the table starts off empty. Whenever a packet is received on the amateur side destined for a non-amateur host, an entry is made in the table, enabling the non-amateur host to send packets in the other direction. After a certain period of time, these entries time out if packets have not been received from the amateur side of the gateway.

This scheme can be augmented with a few new ICMP messages. One message can force an entry to be removed from the table of authorized non-amateur systems. This allows the amateur radio operator that initiated the link to exercise his control operator function to cut off the link if he detects inappropriate use. Another message would allow one to add an authorized non-amateur host to the tables with an appropriate time to live. Both these message are allowed to come from either side of the gateway, but if they come from the non-amateur side, they must include a call sign and a password of for an authorized control operator for the gateway.

7. Status

The packet radio implementation of IP works. We have successfully connected from an **IBM** PC with a packet radio controller to a machine on our department's ethernet using telnet.[8] The connection was made using our MicroVax-I as a gateway. We also were able to telnet from the machine on the ethernet to the PC.

In the Seattle area, we are using a duplex repeater as the base for our local area network. Our network extends from Seattle, south to Tacoma, west to a station on the other side of Puget sound, and east to the Cascades.

We have not yet written the user program to support connection mode logins, but that is being considered. We also have not yet done anything towards using NET/ROM to interconnect our local area networks with others, but we would like to do that soon.

8. Conclusions

The Unix operating system provides a nice base upon which network services can be provided for the amateur packet radio community. At the same time, such a system can serve as a central node in the interconnection of local area networks running **IP,** and even those that don't run **IP.** By linking packet radio networks with more established networks, additional services become available. Such services are available in the Seattle area. These services are necessary if we are to interest people in running TCP/IP. Further, interconnection with non-IP packet radio users is necessary if we are to interest users who would like to try **IP,** but still want to maintain connectivity with those still using connection mode.

[8] One of several remote login protocols.

9. Acknowledgments

A number of people were helpful in getting our implementation running and in discussing some of the ideas presented in this paper. Among them are Bob Albrightson (N7AKR), Bob Donnell (KD7NM), Dennis Goodwin (KB7DZ), Mike Chepponis (K3MC), Steve Ward (W1GOH), and Ed Lazowska (KG7K). Thanks are also in order to Bob Hoffman (N3CVL) for typesetting this paper.

10. Bibliography

1. Fox, Terry L.: AX.25 Amateur Packet-Radio Link-Layer Protocol. Version 2.0. American Radio Relay league, October 1984.

2. Karn, Phil: "TCP/IP, A Proposal for Amateur Packet Radio Levels 3 and 4", Fourth *ARRL* Computer *Networking Conference,* San Francisco, 1985.

3. Leffler, S; Joy W.; Fabry R.; Karels M.: Networking Implementation Notes 4.3 BSD Edition. Computer Systems Research Group, University of California, Berkeley. June 1986.

PACGRAM MESSAGING PROTOCOL FOR PACKET NETWORKS

Jay Nugent, WB8TKL
3081 Braeburn Circle
Ann Arbor, Michigan 48108

In the 5th Computer Networking Conference papers David Cheek, WA5MWD, included in his submission an attatchment containing the Pacgram Protocol Definition. In this paper you will find the latest revision of that Pacgram Protocol Definition with a synopsis of Pacgram's creation, where it is at now, and where I hope it will go in the amateur packet radio community.

BEGINNINGS

Pacgram's beginnings date back to the good old days of the BETA board tests. It was recognized early on that we were going to require some standard method of passing Radiogram traffic via packet radio. Though the Radiogram form has been in use for many years, the way in which it is used varies from one network to another and from one mode to another. In this I mean that each network has its own particular way of presenting the contents of the Radiogram over the air for a receiving station to copy.

Each network has developed a method that best suits its needs based on the particularities of the network. Voice traffic nets present the Radiogram contents by reading the Radiogram slowly and in chunks or logical groups, stopping at predefined breakpoints for fills or varification. CW traffic networks have a similar method varying slightly to suit the needs of the CW operators network.

With this in mind it only made sense that the packet radio network, with its totally different conditions, would need to develop its own method of presenting Radiogram traffic. To this brings us the need for a set of objectives to be strived for cr met when designing our new protocol.

OBJECTIVES

0 Provide a 'standard'

The most important of all is to design a messaging protocol that would be the standard that all packet radio traffic would follow. To accomplish this would require that the protocol suit all the needs of the traffic handelers.

0 User readable without software

Since the equipment and capabilities of each packet radio station varies, it would be desirable that the protocol use only printable characters so that traffic handelers that did not have Pacgram software would still be able to operate within the network.

Though a little more difficult when originating traffic, it would be a simple task 'when receiving or forwarding a Pacgram to decode the fields by hand or simply receive the message to disk for later re-transmission.

0 Compatable with other networks

Any traffic handler knows that a piece of traffic can travel through any number of different nets before reaching its destination. These nets may be Packet, RTTY, CW, or Voice traffic networks.

This requires us to keep in mind any limitations that these other networks may have when we are originating and even when forwarding traffic. Pacgram must be designed to filter out any non-printable characters along with a number of the computer keyboard characters that cannot be sent in CW or RTTY modes.

0 Not machine dependent

The Pacgram messaging protocol should not have any machine dependancies. The protocol should not require a specific type of terminal, computer, hardware, or screen format. A session layer protocol can be implemented at each Pacgram station to handle these dependancies. This session layer has been implemented in the current CP/M version of Pacgram to support ADM3 style consoles like the Xerox-820.

- Reduce operator workload

An important gain in utilizing computers is to allow the computer to do as much of the tedious work as possible. When implementing Pacgram we should have the software perform automatic word counting, logging, filing, etc.

(Please refer to David Cheeks paper in the 5th Computer Networking Conference on page 5.115.)

- Simple

The 'KISS' principal seems to apply to everything. An important objective is to keep the Pacgram protocol as simple as possible without giving up important options such as future growth and expansion.

- Forwardable

Since a message forwarding network is already in place we should maintain compatability with it. We should also keep in mind future network designs. Pacgram being simply a string of text characters allows us to upload, download, and make use of the automatic mail forwarding features of the WORLI type of Packet Bulletin Board Systems.

Maybe future PBBS's could contain Pacgram pre and post-proccessors for users that did not have Pacgram software, This would allow them to originate and receive Radiogram traffic while the backbone network utilized the Pacgram protocol.

- Expandable

Pacgram's design should include the ability to be expanded and adopted into other applications. One of these applications may be a software interface that extracts Pacgram fields and inserts them into a database,

As you will see, the latest revision of the Pacgram protocol allows for formtypes other than the ARRL Radiogram so database applications can be extremely flexible.

- Automatic or Unattended

Pacgram implementations have been written that allow for automatic receipt of traffic. This can be extremely handy when the operator cannot be at the terminal at all times such as during an emergency situation. Other implementations could allow automatic printing of all Radiogram traffic to hardcopy as they are received while at the same time saving all other packet activities to disk.

CODING THE PROGRAM

With these objectives in mind I started writing a terminal program that contained the Pacgram messaging protocol. The project started out being written in BASIC since nearly every computer has a BASIC interpreter and that would allow a greater number of machines to support Pacgram. But this failed miserably when it was determined that BASIC simply was not fast enough without being compiled, and this certainly would limit the number of users due to the lack of BASIC compilers for many of the smaller computers.

This lead me to write the Pacgram Terminal Program in 8080 assemoly code to run under CP/M. This would limit the use of Pacgram to only CP/M machines and others that could emulate CP/M such as the IBM-PC, Apple, and some Commodores.

Programming started in February 1984 with software tests both in the Detroit and Dallas areas, followed by many re-writes and updates. Refinements were made to the protocol as problem areas were uncovered.

IMPROVEMENTS

Recent improvements have been to seperate the Area Code into its own field to allow for destination sorting based on Area Codes. The addition of the Formtype field to identify the type of form contained within the Pacgram. This allows us to not only send ARRL Radiogram traffic but Civil Defense traffic as well. Not to mention other formtypes that may be required for special events such as Footraces and Bikethons and databases.

And finally, the addition (but not yet implemented) Application Level Acknowledgement. Since our only acknowledgements are at the Link layer, there was no way of guarenteeing that the Pacgram was properly captured by the receiving station so an additional verification is being considered.

For more details on the Pacgram protocol, see the Pacgram Protocol Definition that follows at the end of this paper.

FUTURE

It is hoped that Pacgram will eventually catch on as the standard for Radiogram, Emergency, and/or Database messaging. The protocol is still developing but appears to be quite dependable and stable at its current revision.

As for the future, good news. In the ARRL FIELD FORUM dated January 1987, after a meeting of the Eastern Area Staff of the National Traffic System in Williamsburg, Virginia, the following resolution was published:

"THEREFORE, the Eastern Area Staff RECOMMENDS that the Eastern Area Staff Packet Committee continue its work towards defining a strategic traffic system which integrates packet, and

FURTHERMORE, the Eastern Area Staff will implement, in the near term, the following National Traffic System usage of packet for a trial period of two years:"

"...and. FURTHERMORE, the Eastern Area Staff recommends the use of the PACGRAM message format, as described by the 5th Networking Conference, during the trial period. "

It is hopeful1 that the NTS Eastern Area Staff will use the most current revision of the Pacgram message format as published below.

[Pacgram Protocol Definition]
(Versions 2.0.3 and later)

By: Jay Nugent WB8TKL
3081 Braeburn Circle
Ann Arbor, Michigan 48108
(313) 9714076

Dated: 860813

PACGRAM is an application software package that runs on the host computer connected to a TNC. The PACGRAM software is responsible for prompting the operator for the proper Radiogram information one field at a time and forms a PACGRAM message from this. The message can then be sent to the TNC for transmission into the amateur packet network.

On the receiving end, PACGRAM decodes the data stream from the TNC for the starting characters of a PACGRAM. When it finds these characters it receives the rest of the message and can later decode it back into the Radiogram format for display on the console, or printed on the printer. Received PACGRAMS are stored in buffer space and/or disk files and may be later retransmitted or forwarded to other stations in the network.

Special characters are used within the PACGRAM to signal the start of the PACGRAM message, the end of the PACGRAM message, and to seperate the fields of information within the PACGRAM message.

These control characters and the protocol are described in the following definition.

----- PACGRAM CONTROL CHARACTER and PROTOCOL DEFINITION

The control characters, and character sequences, used in PACGRAM were based on the unlikelihood that they would ever appear in any part of a normal Radiogram.. Consideration of the CW traffic nets that may handle message traffic generated with PACGRAM was also taken into account since many characters cannot be sent using CW.

For those stations not possesing PACGRAM software, these control characters were selected so that a PACGRAM can be read directly from a terminal and written back into the standard Radiogram form very easily by hand. A Formsmode has been added to the protocol to allow the sending of a directly printable PACGRAM. This enables any station to receive a PACGRAM already formatted to be printed on hardcopy. This form of PACGRAM uses its own starting sequence that can be easily detected by a small computer running BASIC. Once the start sequence is detected, it can then route all output to the printer. The standard end of PACGRAM character is used to indicate the end of the print so that the output can then be routed back to the console.

---- The START of a PACGRAM shall be a pound sign '#' followed by an asterisk '*'.

The purpose of this character sequence is to signal the start of a PACGRAM and differentiate it from any other data sent by the TNC to the host computer. Such as other communications data or commands and responses from the TNC.
The start of the PACGRAM sequence was altered in the second release of this protocol in an effort to avoid false starts caused by WORLI like PBBS's that use the # character in their data.

---- The FORMTYPE sequence shall be 'ARL' for ARRL Radiogram format and shall always be three characters in length followed by the DELINIATION character.

The purpose of this three letter sequence is to identify the form type that is contained within the Pacgram. For the standard ARRL Radiogram form this sequence is set to ARL. Applications requiring message formats other than ARL may define their own field names and order and define their own three letter Formtype.
(This change has been made for releases 2.0.0 and above)

---- The DELINIATION character shall be an asterisk '*'.

This character is present in the PACGRAM to deliniate the Radiogram fields from one another. The absense of data in any one of the fields will cause two consecutive asterisks to appear within the PACGRAM. No filler characters are placed between the asterisks of an empty field.

---- The FIELD ORDER of an ARL PACGRAM is as follows.

NUMBER / PRECEDENCE / HANDELING INSTRUCTIONS / STATION OF ORIGIN CHECK / PLACE OF ORIGIN / TIME FILED / DATE FILED

NAME TO / NUMBER & STREET / CITY / STATE / ZIP / AREA CODE / PHONE NUMBER

TEXT OF THE MESSAGE

SIGNATURE / TITLE OF SIGNEE

Note: The CITY and STATE fields have been seperated into two individual fields. This allows for automated routing based on the destination City and State. Area Code was seperated for the same reason.

---- FIELD LENGTHS and TYPES within the ARL PACGRAM.

The Number field will be limited to 8 characters maximum.
The Check field will be limited to 10 characters maximum.
(This has been expanded to handle ARL type checks)
The Text field is limited to a maximum of 1024 characters.
All other fields are limited to 60 characters.

Fields may contain either alphabetic or numeric characters but those characters that cannot be sent using CW will not be allowed.

Such as the following:
- Pound sign * - Asterisk & - Amperand
% - Percent < - Left arrow > - Right arrow
= - Equals ^ - Carat - Underscore
All control characters including Carriage Return, Linefeed, and Formfeed.
Any non-allowed characters found within a Pacgram will be discarded.

---- The END of a PACGRAM shall be the ampre sign '&'.

The purpose of this character is to signal the end of the PACGRAM.

---- The START of a FORMSMODE PACGRAM shall be '#PAC*' followed by a carriage return.

The purpose of this starting sequence (including the carriage return) is to signal the start of a PACGRAM in the FORMSMODE. A small computer running a simple BASIC program can trap this starting sequence and then direct all its output to a printer.

---- The END of a FORMSMODE PACGRAM shall be an ampre sign '&' followed by a carriage return.

The purpose of the end of FORMSMODE sequence is to signal the end of a FORMSMODE PACGRAM. A small computer running a BASIC program can trap this sequence and then direct its output back to the console.

---- Proposed Application Level Acknowledgement shall be as follows:
#*ACK*Number*Precedence*HX*Station of Origin&

The purpose of the Application Level Acknowledgement is to confirm to the sending station, in a positive manner, that a Pacgram sent by it to another station has in fact been received by the distant station.
Note the Startpac sequence followed by ACK to indicate acknowledgement. The NUMBER, PRECEDENCE, HX, and STATION fields are echoed back by the receiving station for comparison to SENT Pacgrams. A match confirms which SENT Pacgram has been acknowledged. It can then be flagged as ACKnowledeged and logged.
The NUMBER and STATION fields should never be repeated within any operating year so the likelihood of an incorrect ACK is very unlikely.

==========

As a Radiogram has well defined fields in a specific order, so does the PACGRAM. The order in which the fields occur in the Radiogram is the exact order that they will appear in the ARL PACGRAM. For example, here is a sample Radiogram followed by its equivelent data stream in PACGRAM form.

NUMBER: 126 ROUTINE WB8TKL CHECK: 5 ANN ARBOR MI 14302 MAY 21
TO: MIKE NUGENT
123 HOLLYWOOD AVE
HOLLYWOOD, CAL 54321
(818)555-1234
TEXT: HOW IS THE WEATHER X
SIGNED: JAY

And the equivelent in ARL PACGRAM form would be:

#*ARL*126*R**WB8TKL*5*ANN ARBOR MI*1430Z*0521*MIKENUGENT*123 HOLLYWOOD AVE
*HOLLYWOOD*CAL*54321*818*555-1234*HOW ISTHEWEATHERX*JAY*&

The proposed Application Level Acknowledgement to this would be:

#*ACK*126*R**WB8TKL&

As you can see, the length is well within 256 bytes, tne maximum AX.25 packet length. Even with the Paclen set to 256, a single packet could contain a fairly large text field. Shorter Paclens can be used as network conditions require. The Pacgram will simply be re-assembled back into its original form frame by frame.

You can see the benefits of using PACGRAMs over the voice or CW methods of sending traffic. This packet can be sent in just a fraction over one second at 1200 bps, an enormous improvement over existing amateur traffic systems. Pacgram also prompts the operator for the fields in the correct order, so fewer mistakes are made.

Also notice in my example, that since I left the fields for Handling Instructions and Title blank, that PACGRAM simply put no data between the two asterisks. This is neccessary to maintain the field count for decoding of the PACGRAM at the receiving end and also wastes as little of the transmission bandwidth as possible.

Happy Packet ing -WB8TKL

Performance Monitoring
-or-
"I Wanna Fix It, Is It Broke?"

Skip Hansen, WB6YMH
Harold Price, NK6K

Abstract

Much of the performance informati.on on Amateur Packet Radio is anecdotal and ephemeral; a subjective and non-detailed accou.nt usually limited to a gross statement of "goodness" or "badness", which is neither well documented nor long remembered. While there are several papers which describe the expected performance of CSMA-type systems, there is little actual data about the live amateur packet system.

The authors discuss the need for accumulating performance data and describe work in progress to supply performance measurement software using a C program and a TNC with KISS software.

1. Why Performance Monitoring?

Big changes are coming in amateur packet radio. In early 1987, most of the amateur packet network was based exclusively on AX.25 and digipeaters. By the end of 1988, if not sooner, much of the packet world will be made of up a conglomeration of NET/ROM, TEXNET, TCP/IP, and other systems interconnecting 40,000 AX.25 based users. Each will be implemented and installed by packeteers eager to make the network better than it was before.

Each system contains a myriad of trade-offs and compromises. Each system has several tuning knobs which can be used to modify the way it operates, affecting both local user performance and global network performance. In many cases, these knobs will be cranked by people with no data on how things are running and therefore no way to tell if anything got better. In other cases, the knobs will be tuned to optimize local performance, to the undetected detriment of the rest of the network.

Performance data is vital to a local network. It is needed before the current network can be tuned, and it should be available to those who will help specify the next network. Put simply, if you don't know what you have now, how will you know if what you get next is any better?

1.1 We've Already Missed One Chance.

We've already missed one chance to monitor a major change, and in California, we've missed a second. The first version of AX.25 did not use the Poll/Final facility of LAPB. In that version, if an acknowledgment of a data frame was not received, the data frame was re-transmitted. If multiple data frames were outstanding, only the first one was re-sent. In the second version of AX.25, the poll/final facility was implemented. In AX25v2, if a data frame is not acknowledged, a "poll" is sent out, soliciting a new acknowledgment. If that ack does not indicate that the data frame was received, the data frame is then retransmitted, otherwise transmission continues with new data frames.

Any change to a protocol like the one described above entails some cost. Whether it is the effort involved in updating and distributing new software, or the trek to a snowed in mountaintop to swap ROMS, some of our limited people resources are expended. In the poll/final update, was a improvement in network performance obtained that in some way offset the effort involved in implementing it and updating the user base?

Unfortunately, we'll never know. Since there was no network performance data before the change, and none was taken after, there is no way to tell. Our only indication is indirect; one of the original major proponents of the change to poll/final is now suggesting that poll/final not be used in some cases. [1]

For future changes, we must do better.

1.2 NET/ROM

In California, the old digipeater backbone which connected northern California, southern California, and Arizona has been largely supplanted by NET/ROM nodes. We had no data showing the performance of the old system, and we have no data on the performance of the new system. It is therefore difficult to measure the improvement.

2. Field Experience vs. Theoretical Predications.

There is a large amount of literature on the topic of packet switching systems, and on packet radio. Some is quite accessible to the average amateur, one networking textbook in particular, by Tanenbaum [2], has been cited so often that it is stocked by local amateur radio stores. There is little written, however, on packet as it is practiced in the amateur radio world. In most cases, if you notice the discussion leaning toward the way we do it, you find it given as an example of the wrong way. Actually, the word "wrong" is seldom used, "less optimal" is more common.

Much of the non-amateur networking experience of those who make up the amateur packet radio community is in the area of local area networks (LANs). Although there are a great many common problems and solutions between commercial LANs and the amateur packet network, there is a danger in assuming calculated performance parameters for the former have relevance in the later. Unfortunately, there is a tendency, with a lack of actual data, to use predicted LAN data in design and implementation discussions as if it were gospel.

A LAN, as discussed in Tanenbaum [2] page 286, generally has three distinctive characteristics:

1. A diameter of not more than a few kilometers.

2. A total data rate exceeding 1 Mbps.

3. Ownership by a single organization.

Although (1) is of importance only as it relates to propagation delay for very high data rates, (2) and (3) are worthy of note. The standard data rate in 1987 is still 1200 baud. There are 56kbps modems being beta tested now, but that is still only 6% of 1 Mbps. Ownership by a single organization is also something that is unusual in the amateur radio network. Item (3) tends to lead to either a homogeneous set of network hardware, a common set of goals, or at least a common forum for discussing those items. In the amateur world, users and implementors in northern California, Southern CA, and Arizona don't get together very often. That's another advantage of (1), in Los Angeles, one node can cover a area with a diameter of 200 miles.

Many of the studies done on LAN performance make assumptions that are not valid in the amateur environment. One study, for example, from [2] page 289 assumed:

- All packets are of constant length
- There are no errors, except those caused by collisions.
- There is no capture effect.
- Each station can sense the transmissions of all other stations

None of these assumptions hold for our current environment.

Another difference between our network and more commonly modeled networks is in the large number of autonomous stations we have on the network, and the large number of different traffic patterns running simultaneously. During the three days that data was gathered for this paper, 371 transmitters were on the air at some time in southern California. The peak number of active transmitters on a single 1200 baud frequency in a single five minute interval was 42. Most modeled networks have higher baud rates and/or low data throughput, and assume traffic is moving between a large number of outlying stations and a central station.

Again, while there is much value in reading and modeling, we should make the attempt to measure what we have; both to feed the result back into the models, and to establish a base against which future modifications can be judged.

3. The Current State of Affairs.

There appears to be only one kind of monitoring being done in amateur packet radio today. The two most common BBS systems, by WORLI and by WA7MBL, both produce a log of BBS activities. An analysis program produces a report for the BBS operator of the number of connects from users and the number of messages forwarded, among other items. While this gives a BBS operator some idea of his local usage patterns, it does little to described total network activity, or even the throughput the BBS experiences.

For global network performance we are left with anecdotal evidence, e.g., "01 really stinks tonight" (translation: performance is less than expected), and "I had no problem with 01 today" (translation: I'm retired and was on at 10:00am).

For local user performance, we get "I can talk to Utah all night long", and "I haven't been able to connect up north all week". Obviously, we need something better.

4. It's Not Easy

There are two ways of looking at network performance, one is from the network's point of view, the other is from the user's point of view. In the first case we are interested in how the channel is performing, in the simplest view, how many bytes of data it is carrying. Is the network carrying a large number of user bytes, or is most of the capacity going to overhead or retries? Are we losing data to collisions, or to bad RF paths?

In the second case, the user's point of view, the questions are more toward what level of service an individual user is getting from the network. Is the response time from distant locations adequate? Do many connections time-out? Are some destinations unreachable due to congestions or path failures?

There are several ways of acquiring performance data. One is to have each user station collect it. As updating 40,000 user's is a non-trivial exercise, we've chosen another route. A specialized monitor station sits at a central place and looks at all the activity on the channel. Unfortunately, it isn't easy to answer any of the questions from a third party monitor station. Some of the problems are discussed below.

4.1 The Problem Is, It's Radio.

In most wire based, broadcast-type LANs, a monitor program can make the assumption that if it heard a packet, everyone else in the LAN heard the packet. More importantly, if it didn't hear a packet, no one else did either. Even if the LAN is relaying data between two other LANs, it is at least certain that for data originated on the LAN or destined for the LAN, the monitor has a high probability of having the seen the same data as the other stations on the LAN. In the amateur packet network, due to hidden terminals, the FM capture effect, and propagation, all stations do not hear the same packets.

If the monitor station heard all packets, it could easily follow the state of all connections on the LAN. For connection oriented protocols like AX.25 and TCP, and providing the monitor has been up as long as the other stations on the LAN, the monitor can tell how long a connection has been in place based on the circuit start and end protocols. In the amateur radio case, the monitor station can not be certain that it heard all packets. It may miss a circuit startup or end. It must instead be prepared to infer that a connection exists because it sees data flowing, or that a circuit has closed because it has seen no data for an interval of time. This will add uncertainty to data gathered in an RF environment but it does not invalidate the entire effort.

Although collisions can be directly detected on a wire LAN, they can not be as easily detected on radio. Due to the capture effect, a stronger FM station will completely override a weaker station such that stronger packet is received without error, even though two packets were being transmitted at the same time. A collision may be inferred if the received packet is seen again.

Some tasks then become exercises in gather as much information as possible, and then making an educated guess. Still, this is better than no data at all.

4.2 Users Are Easy to Replace.

It is somewhat easier to gather user oriented data, e.g., does a path to station X exist at this time, or what is the round-trip delay for packets between Los Angeles and Salt Lake City. The monitor station can actually be a user and directly measure these values.

While data can be gathered about the performance of the channel at a specific time in this way, this alone will not supply information about the global network status at the time the measurement was taken. To be able to draw a meaningful conclusion from the data, aside from variable X was equal to Y at time T, other information is needed, such as the number transmitters on the air, and the number of other packets on the channel. In sort, both types of monitoring must be performed, direct measurement of user performance and global network measurement.

5. Monitoring Software

The software currently under development by the authors addresses the problem of global network monitoring. Other types of monitoring will be added in the future.

In this early version of the software, we are attempting to determine what sorts of questions can be answered by a program which listens to a channel and takes note of the packets it hears. Some questions, such has how many total bytes are being received at the monitor site, how many transmitters are seen, how many beacons are heard, are easy to answer.

A much more difficult question is "How many times does the average forwarding BBS send a 20k file before it goes all the way without timing out?" The type of information we're collecting, and the type of questions that can be answered, are discussed below.

5.1 Questions to Answer

There are two basic questions which are reasonably easy to answer. One is "What is the efficiency of the channel", the other is "How many users does the channel support".

We have chosen to define efficiency as the ratio of the number of unique bytes of user data on the channel verses the total number of bytes on the channel. "Unique data bytes" is our term for actual user data not including frame overhead, retransmitted copies, or digipeated copies. For example, if the string "hello" is entered, digipeated once, not acked, retransmitted, redigipeated, and acked, the total number of bytes on the channel would be 168, the number of unique data bytes is 5, an efficiency of 2.9%. If 256 user bytes are sent and directly acked, the efficiency is 88%.

To keep statistics on each user of the channel, we store pairs of Source and Destination calls from the frame header. The pair is called a circuit. A normal two-way connection would consist of two circuits. If NK6K and WB6YMH were connected, one circuit would be (TO:NK6K,FROM:WB6YMH), the other circuit would be (TO:WB6YMH,FROM:NK6K). Statistics for each circuit are maintained separately.

In addition to two basic questions, we wanted to be able to determine the number of digipeaters the circuit used, what the average size of a data frame was, the number of RNR (input blocked) frames transmitted, and similar questions. Since this required looking into the control fields of the frame, the standard TNC interface was unsuitable.

5.2 KISS

We chose the KISS TNC interface to give us access to all fields of the frame. KISS sends the entire frame, minus the checksum, to the terminal port using an async framing format. The KISS interface has been implemented on the TAPR TNC 1, the TAPR TNC 2 and clones, and on the AEA PK-232. The KISS software for the TNC 2 is included with the KA9Q TCP/IP package.

There are no modifications required to the KISS code for use in this application.

5.3 Software Design

The current implementation of the monitor package consists of three programs.

- STATS.EXE - This program monitors the received frames and accumulates data, periodically dumping the data into a log file. STATS also displays the addresses, data, control fields, and a "retry" flag in real-time as frame are received. NET/ROM and TCP/IP control fields are also displayed.
- REPORT.EXE . This program massages selected data from the log file into a form suitable for passing to a plotting program. The plotting program is not included.
- AVERAGE.EXE - This program massages the output of REPORT, combining and averages the records into larger intervals of time. This can result in clearer plots.

5.3.1 STATS.EXE

STATS collects data over a five minute interval, storing it into several different tables. These tables are then written into the log file at the end of each interval, along with a time stamp record. The tables are summarized below.

Digipeater Data.

The total number of packets and bytes heard from a digipeater is stored, along with the call of the digipeater.

Frequency Data.

Totals on bytes and packets heard on the channel without regard to source are maintained. Packet are also counted by length into five buckets: 32, 64, 128, 256, and greater than 256 bytes. The total number of ticks of the 18.2 Hz clock when the data carrier detect (DCD) line was high are recorded, as are the number of ticks when , DCD was low.

Circuit Data.

Several items are stored for each circuit, or TO:/FROM: pair. This includes the number of digipeaters used, the Protocol ID Byte (PID) of the last I frame received in the interval, the total number of packets and bytes received, the number of unique packets and bytes received, and the number of packets and bytes ignoring those heard from multiple digipeaters. Also included is the number of unique frames heard of each frame type (sabm, ua, etc.), the number of frames with POLL, and the number of frames with FINAL. The number of I frames heard is also counted into

five buckets based on the data size: 32, 64, 128, 256, and greater than 256 bytes.

As an indication of the difficulty of accurately determining the status of a frame, the algorithm used to determine uniqueness is described below.

Uniqueness

Depending on the packet type one of three different algorithms are used to test for uniqueness.

I frames are judged to be unique if the N(s) variable matches the expected V(s), or if the locally computed checksum of the information portion of the current frame does not match the checksum of the last frame received with the same N(s). Note that the checksum is only used to resolve the ambiguity resulting from lost frames. An algorithm based solely on checksums would be confused easily by data streams containing identical consecutive lines. For example consider the transmission of text files containing multiple blank lines separating pages. In such cases several consecutive packets would contain identical information, a single carriage return, but still be unique.

S and U type frames are judged to be unique if the control field of the current frame is different than the control field the last S or U frame received. Note that this does not detect retries of frames such as multiple SABMs sent because the target station is not responding.

UI frames are judged to be unique frames if the checksum of the information field of the current frame is different than the checksum of the last UI frame which was received.

Digipeated frame filtering logic

The various "non-digipeated" counters in the software are designed to show the number of times a particular frame appears on the channel without regard to multiple retransmissions by digipeaters. The "non-digipeated" counters are advanced once and only once regardless of how many digipeater hops are observable by the monitoring station. This data is used to determine the number of retries of a packet without confusing a retry for a digipeat.

The software maintains bit maps of observed hops for its use in filtering out digipeated frames. A separate bit map of observed hops is maintained for UI, S and U frames types as well one bit map for each outstanding I frame. There are 9 bits in each map which correspond to the originating station plus up to 8 digipeaters.

A frame is considered to be "non-digipeated" when it is either heard for the first time or it is heard from a hop from which it had been previously heard. The first condition is met when a frame is first transmitted, the second condition is met when a frame is retransmitted successive times. If neither case is met the frame is a digipeated frame and is not used to increment the "non-digipeated" counters.

The digipeat bit map is cleared when either the uniqueness subroutine determines the frame is unique or when the digipeat filter subroutine determines that the frame is a retransmission.

5.3.2 REPORT.EXE

REPORT produces several output formats. The RAW format displays each field in each record. This is useful if a particular interval is being examined in detail, or when debugging STATS. Several other formats are used to produce data for plotting. One report totals all circuit data for an interval. Examples of this output are provided later.

5.4 Hardware

As discussed above in the section on KISS, TNC 1 and TNC 2 clones, and the AEA PK-232 can be used with this software. If the DCD ON and OFF times are desired, a jumper must be added.

DCD Jumpering

Since most of the current TNC designs use the DCD signal on the RS-232 interface as a connect status indicator it is necessary to modify the TNC hardware slightly to provide a true modem DCD on the RS-232 interface. The modification for the TNC 2 and clones is very simple, consisting of a single jumper wire. The jumper goes between pin 2 of the modem disconnect header (DCD output from the modem) and the pin of JMP1 which is NOT connected to +5 volts (input to the DCD driver). On the MFJ-1270B artwork the correct pin of JMP1 is the one closest to the front panel. The authors have not researched modifications to other TNC designs, but it is expected the modifications will be similar. It is NOT necessary to perform the DCD modification to run the monitoring software, it is only necessary if the statistics of DCD activity are desired.

Most terminal software used on packet will be unaffected by this modification, however most BBS software will require the jumper to be removed for normal operation.

The software was developed on an IBM PC/AT using Microsoft C 4.0. It should be easily transportable to other systems provided a suitable serial port interface is available.

A hard disk is highly recommended. Twenty-four hours of data for 145.01 MHz as monitored in southern California produced 500k bytes of log file data. This may be reduced, of course, by increasing the interval time.

6. Examples

We used the STATS program to acquire performance data on all of the active packet channels in southern California. The monitoring site used for 145.01 MHz was at 700 feet on Palos Verdes. On this frequency, the site can "see" 8 NET/ROM nodes. During the 24 hours during which 145.01 was monitored, from 00:00 to 23:59 local time on a Thursday, 105 total transmitters were seen.

The data shown in the sample graphs is based on the five minute interval data from STATS, which was then processed by REPORT. The output of REPORT was then averaged by AVERAGE into 15 minute samples. Each point plotted represents the average of three five minute intervals.

Figure 1 shows the number of user circuits seen in an interval. A "user circuit" is a subset of the total circuits; beacons, repeater ids, and other circuits consisting of a small number of UI frames have been removed. The peak of data just after midnight is caused by forwarded BBS traffic, large broadcast messages such as newsletters are restricted to being forwarded between 00:00 and 8:00 by local custom.

Figure 2 shows the total bytes per minute. Further analysis of the data would show the distribution of bytes in the peaks; how much is destined for local users, and how much is going "overhead", passing through the backbone to other NET/ROM locations. The major NET/ROM path through to Arizona is still on 145.01, this should change before the summer is out. It will be interesting to see what effect moving the backbone will have on this graph.

Figure 3 shows the efficiency of the channel, computed as discussed above.

Figure 4 is a plot of efficiency vs. the number of user circuits on the channel. The distribution on a plot of efficiency vs. total packets is similar to this one. The occurrence of low efficiency over the entire range of users (and number of packets) shows that there are causes of low efficiency other than congestion. One interpretation would be that more data is lost due to poor RF paths than to collisions. Another would be that hidden terminals are causing problems. Further analysis of the data, coupled with a knowledge of the geography and stations involved, might result in information that could be used to improve the network.

7. Other Uses / Future Goals

Once a basic set of data gathering tools and formats has been define, the applications are boundless. For example:, STATS can be used to make improvements to the current 14.109 HF forwarding scheme. For example, data. gathered during the day on Friday, July 29, shows that of the two top stations in. terms of the total bytes transmitted, shows that one had 30% better efficiency than the other. If monitoring was continued, and the trend continued over time, it may mean that the less efficient station is trying to reach stations beyond its range, or that there are local receiver problems. It also means that the monitor station was hearing more data frames from the transmitting station that the target station was, perhaps the mail between those stations should be re-routed.

STATS can be used to check propagation between the monitor station and other stations. Figure 5 shows then number of bytes received on 14.109 MHz in a 24 hour period. It can also be used to infer propagation between other station. For example, if you hear a station in Indiana sending packets to Seattle and the efficiency is high, then a path must exist between those two points, even if you do not hear packets from Seattle at the monitor site.

The "unique" subroutine can be used filter retries out of a monitored connection as the data of the connection is displayed. AEA offers a similar feature on some of its TNCs. STATS will be updated in the future to allow the capture of filtered text from each circuit into files for later review. This can serve several purposes, as a diagnostic aid, a periodic check for intruders on the amateur network as required by the FCC, or to satisfy the standard urge to "read the mail".

This type of data collection could also assist in message traffic analysis, e.g., how many bytes are in the average connection? Are most of the BBS messages forwarded on a channel destined for users in the local area or are they just passing through?

Currently, STATS monitors at the link-layer level. Higher layer protocols such as TCP/IP and NET/ROM add additional complications to traffic analysis, primarily in determining the actual

origination and destination point. Work remains to be done in this area.

8. Conclusion

There is much good to be gained from gathering and analyzing performance data. It can tell us where we are and suggest where we might go. It will also help determine if we like where we've gone once we get there. The work discussed here is a start toward developing tools to aid in this task. Others are invited to participate.

9. Availability

The software described in this paper is available in source form from the WB6YMH-2 BBS on 145.36 in southern California. This BBS is also available by phone for those not in the local area at (213) 541-2503. Updates will periodically be sent to the HAMNET BBS on Compuserve.

10. Acknowledgments

Thanks to Craig Robins, WB6FVC, for his help in the preparation of this paper. Thanks also to those who have implemented the KISS code for the TNC 1 and TNC 2, and the folks at AEA.

11. References.

[1] Karn, P., KA9Q, "Proposed Changes to AX.25 Level 2", informal paper circulated on various mail systems and reprinted in the July/August 1987 NEPRA PacketEar, the newsletter of the New England Packet Radio Association.

[2] Tanenbaum, A., "Computer Networks", Englewood Cliffs, NJ: Prentice Hall, 1981.

Fig 1. User Circuits

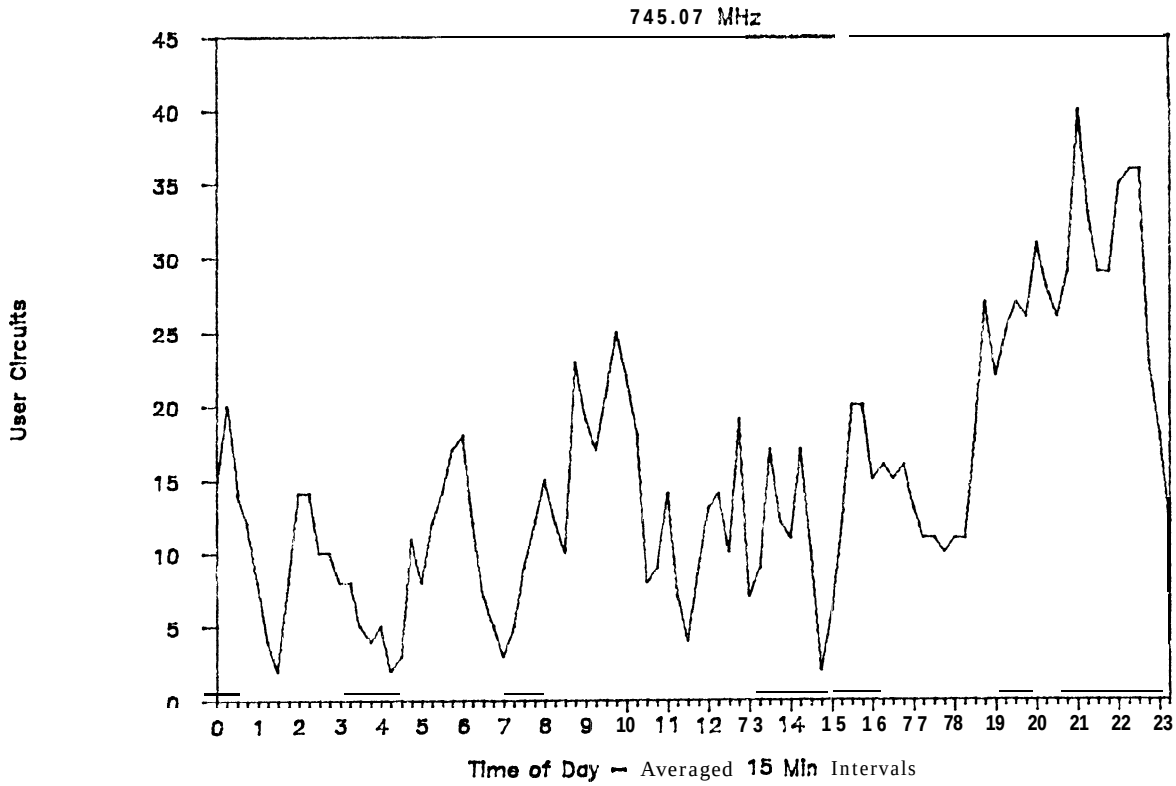


Fig 2. Total **Bytes per Minute**

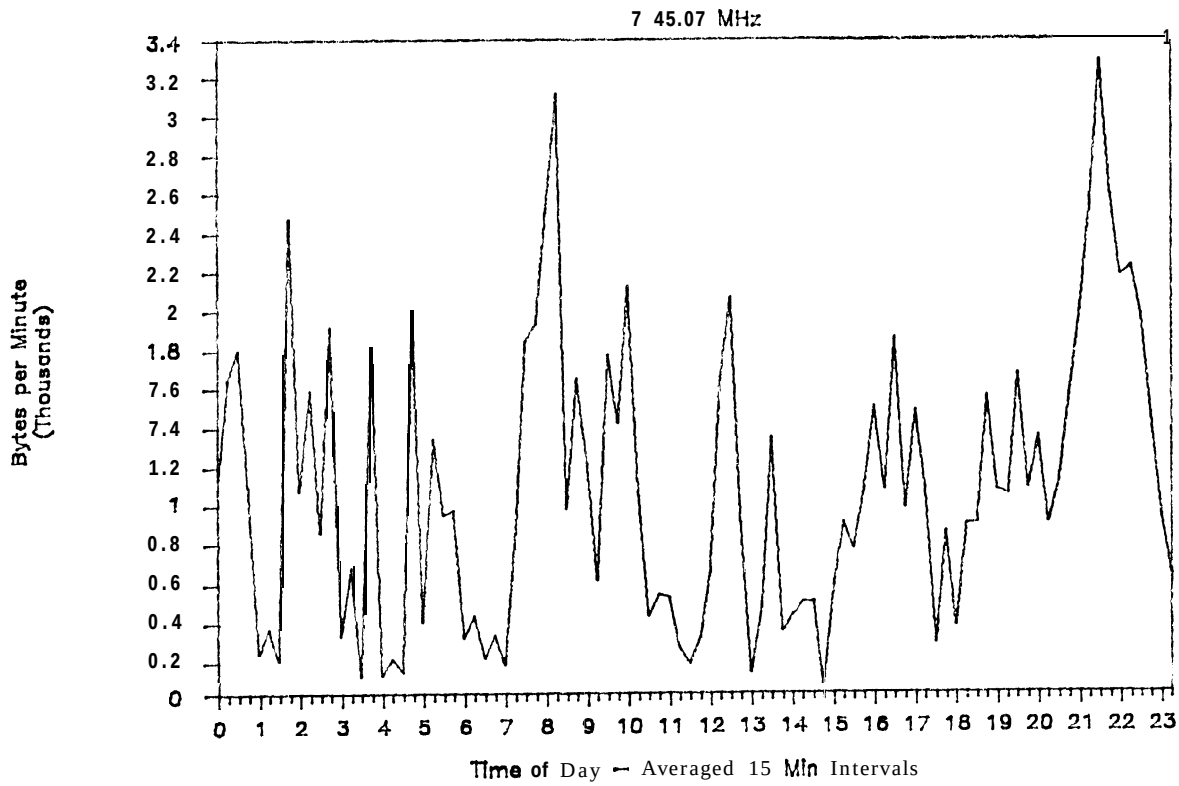


Fig 3. Efficiency

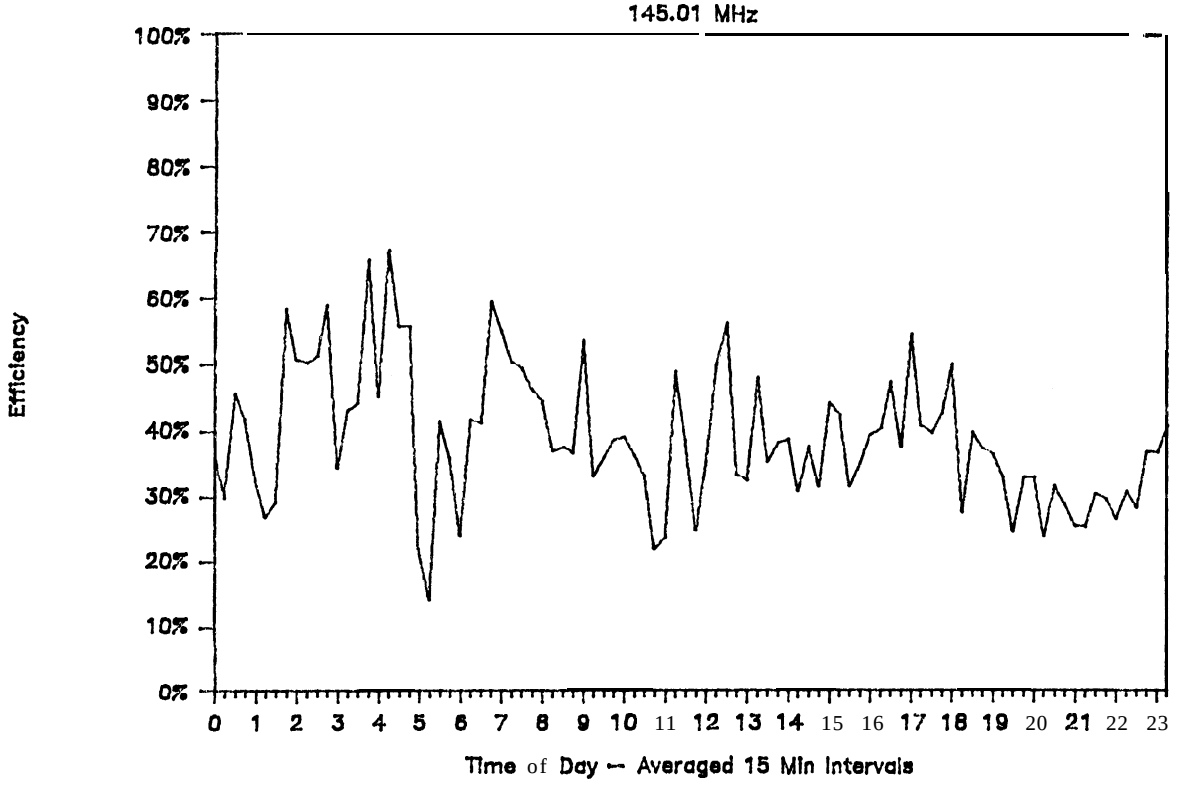


Fig 4. Efficiency Vs. Users

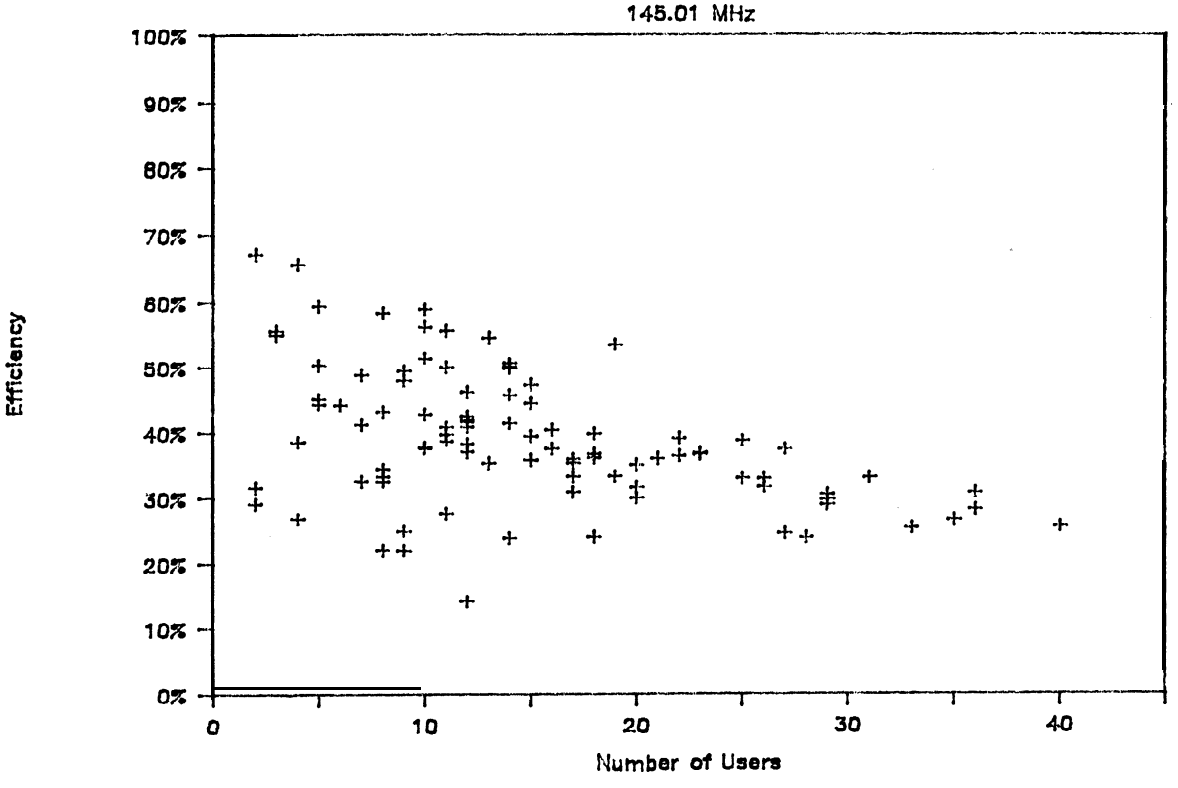
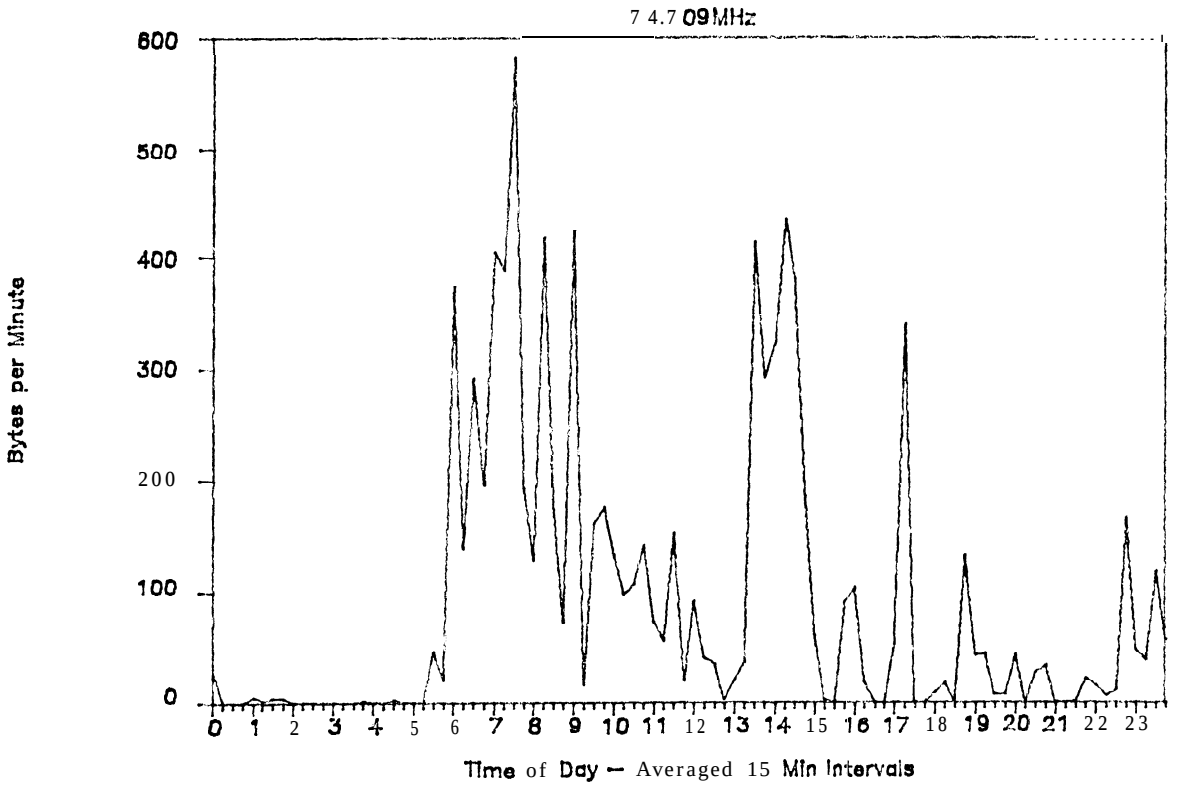


Fig 5. Total Bytes per Minute



DRAFT PROPOSED
AMERICAN NATIONAL STANDARD FOR ELECTRONIC BUSINESS
DATA INTERCHANGE AMATEUP RADIO MESSAGE TRANSACTION SET

Jack Sanders, NC4E

SECRETARIAT

Transportation Data Coordinating Committee

ABSTRACT

This standard contains the format and data content of the Amateur Radio Message Transaction Set for use within an Electronic Business Data Interchange (EBDI) environment.

The ASC Xl2 family of Electronic Business Data Interchange standards are based on interdependency. Several of the ASC Xl2 standards define the data elements, data segments, control structures and acknowledgments that relate to transaction set standards. Availability of the following standards is required in order to interpret, understand, and use the ASC Xl2 family of standards.

ANSI/ASC X12.3 Data Element Dictionary
ANSI/ASC X12.22 Data Segment Directory
ANSI/ASC X12.5 Interchange Control Structure
ANSI/ASC X12.6 Application Control Structure
ANSI/ASC X12.20 Functional Acknowledgment (997)
ANSI/ASC X12.21 Interchange Acknowledgment

Information as to the source of the documents noted above can be obtained from:

Secretariat, ASC Xl2
c/o E. A. Guilbert
TDCC
1101 17th Street NW
Washington, DC. 20036

CONTENTS

AMERICAN NATIONAL STANDARD
FOR ELECTRONIC BUSINESS DATA INTERCHANGE
AMATEUR RADIO MESSAGE TRANSACTION SET

A.l. Purpose and Scope. This standard provides the standardized format and establishes the data contents of an Amateur Radio Message Transaction Set within the context of an electronic business data interchange (EBDI) environment.

A.2. Terms and Definitions. The ASC Xl2 family of Electronic Business Data Interchange standards are based on interdependency. Several of the ASC Xl2 standards define the data elements, data segments, control structures and acknowledgments that relate to transaction set standards. Availability of the following standards is required in order to interpret, understand, and use the ASC Xl2 family of standards.

ANSI/ASC X12.3 Data Element Dictionary
ANSI/ASC X12.22 Data Segment Directory
ANSI/ASC X12.5 Interchange Control Structure
ANSI/ASC X12.6 Application Control Structure
ANSI/ASC X12.20 Functional Acknowledgment (997)
ANSI/ASC X12.21 Interchange Acknowledgment
ANSI/ASC X12.6 American National Standard for Electronic

Business Data Interchange - Application Control Structure, contains the technical definitions of all terms related to Electronic Business Data Interchange. The definitions below are consistent with those formal definitions and are provided here, in shortened form, to aid in the understanding of this standard.

transaction set. A transaction set is composed of the specific group of data segments which represent a common business document--for example, a purchase order or an invioce. A transaction set is the collection of data that is exchanged in order to convey meaning between the parties engaged in electronic business data interchange. Each transaction set starts with a transaction set header, and is immediately followed by a beginning data segment unique to that transaction set type. The transaction set is terminated (ended) by a transaction set trailer.

data segment. A data segment is the intermediate unit of information in a transaction set. Data segments consist of logically related data elements in a defined sequence. Data segments have a predefined data segment identifier which comprises the first characters of the data segment. When data segments are combined to form a transaction set their relationship to the transaction set is defined by a requirement designator and a data segment sequence. Some data segments may be repeated, and groups of data segments may be repeated as loop.

data segment identifier. Each data segment has a unique identifier composed of upper case letters and digits with a length of two or three characters. The identifier serves as a name for the data segment and occupies the first character positions of the data segment. The data segment identifier is not a data element.

data segment requirement designator. A data segment has one of three requirement designators defining its need to appear within the transaction set. The requirement designators are listed below with each followed by the code in parentheses.

Mandatory (M). This segment must appear in the transaction set.

Optional (0). The appearance of this segment is at the option of the sending party or may based on the mutual agreement of the interchange parties.

Floating (F). This designator is used for OPTIONAL data segments that **may** appear anywhere in the transaction set after the beginning segment and before the transaction set trailer.

data segment sequence. Each data segment has a specific sequence within the transaction set. Data segments must appear in this order, except **"F"** designated segments, which may appear anywhere within the transaction set, after the beginning segment and before the transaction set trailer. Data segments may appear in any of the three areas of the transaction set, as indicated below.

+ Heading Area. When a data segment appears in this area, it refers to the entire transaction set.

+ Detail Area. When a data segment appears in this area, it refers to that detail information only, and will override **any** similar specification in the heading area.

+ Summary Area. Data **segments** in this area contain only control totals or actions performed on those totals, such as overall discounts.

maximum use of segments. Some data segments may be repeated multiple times at their specific location in the transaction set. The term **"Maximum Use"** in 3.2, 3.3, and 3.4 refers to the maximum number of times a segment is performed to appear, in succession, at that specific location.

loops of data segments. Within transaction sets specific groups of logically related data segments always appear together. These segment groups are refered to as loops. The term **"Loop** ID/Repeat Count" in 3.2, 3.3, and 3.4 refers to the position and nesting of loops and the number of times each loop is permitted to occur at that specific location in the transaction. One loop may be nested within another loop provided an inner loop terminates before any outer loop terminates.

monetary values. The monetary values that **may** appear in certain data segments reflect the currency of the country of the transaction set originator unless otherwise specified by the use of the optional **"CUR"** segment within a transaction set, (The **"CUR"** segment provides the capability to specify the currency of other countries or to convert the currency of any country to the currency of any other currency).

functional group identifier. Each transaction set is included in a specific collection of similar transaction sets called a functional group, as defined in ANSI/ASC X12.6. The allowable functional identifier for the Amateur Radio Message Transaction Set is **"QNU"**.

A.3. Transaction Set Specifications.

A.3.1 Introduction. The transaction set specifications are presented in 3.2, 3.3, and 3.4. The specifications define the sequence of data **segments**, the requirement designators, maxi-mum use, and loop ID/repeat counts for each of the three areas of the transaction set. Also included are explanatory ***comments*** that relate to the use of certain segments and loops.

A.3.2 HEADING AREA

DATA SEGMENT' SEQUENCE FOR THE HEADING AREA
AMATEUR RADIO MESSAGE TRANSACTION SET

SEGMENT IDENTIFIER	TITLE	REQUIREMENT DESIGNATOR	MAX USE	LOOP ID/ REPEAT COUNT
ST	Transaction Set Header	M	1	
QNU	Beginning Segment (Amateur Radio Message)	M	1	
QPA	Preamble	M	1	
QAD	Address	M	1	

A.3.2 DETAIL AREA

DATA SEGMENT SEQUENCE FOR THE DETAIL AREA
AMATEUR RADIO MESSAGE TRANSACTION SET

SEGMENT IDENTIFIER	TITLE	REQUIREMENT DESIGNATOR	MAX USE	LOOE' ID/ REPEAT COUNT
QTX	Text	M	99	
QSG	Signature	M	1	
QNB	Relay Identification	0	1	

A.3.4 SUMMARY AREA

DATA SEGMENT SEQUENCE FOR THE SUMMARY AREA
AMATEUR RADIO MESSAGE TRANSACTION SET

SEGMENT IDENTIFIER	TITLE	REQUIREMENT DESIGNATOR	MAX USE	LOOP ID/ REPEAT COUNT
SE	Transaction Set Trailer (End)	M	1	

APPENDIX (This Appendix is not a part of American National Standard ANSI/ASC X12.A-1985)

A.A Appendix A
Example Amateur Radio Message Transaction

This appendix contains an example of an amateur radio message document that conforms to the requirements of ANSI/ASC X12.A-1985. Figure Al shows the original radiogram document. Figure A2 shows the data segments that translate those information units to conform to ANSI/ASC X12.A-1985.

The amateur radio message example in Figure A2 does not illustrate the use of all the elements that make up the amateur radio message transaction set, it is only intended to show how a simple radiogram document is encoded to conform to this standard. For more complex documents the use of additional optional elements shown in the data segment diagrams may be required.

AMATEUR MESSAGE FORM

THE AMERICAN RADIO RELAY LEAGUE
RADIOGRAM
via amateur radio

NUMBER	PRECEDENCE	HX	STATION OF ORIGIN	CHECK	PLACE OF ORIGIN	TIME FILED	DATE
1	R	B24	W1AW	8	NEWINGTON CONN	18302	Jul1

TO
DONALD SMITH
164 EAST SIXTH AVE
NORTH RIVER CITY MO 00789
733 4968

HAPPY BIRTHDAY X SEE YOU SOON X LOVE

DIANA

96 NUMBER OF INCLUDED SEGMENTS

AMATEUR PACKET MESSAGE

```
ST*QNU*0008
QNU*R*00789***W1AW*1
QPA*1*R*HXB24*W1AW*8*NEWINGTON CONN*1830*850701
QAD**DONALD SMITH***1645 EAST SIXTH AVE*NORTH R
  IVER CITY*MO*US*00789*7334968
QTX*HAPPY BIRTHDAY X SEE YOU SOON X LOVE
QSG**DIANA********
QNB*ORIGINATE*860808*1441*NC4E*861217*2230*ORIGINATED by W1AW Newington, CT
SE*8*0008
```

3. Data Element Dictionary.

3.1 Introduction. The data element specifications are presented in 3.2. In addition to the specifications and formal definitions, this standard also contains cross-reference information to the appendixes. The data elements are listed in data element reference number sequence in the standard.

Some data elements contain references to either the Appendix A Code Sources or the Appendix B code Lists. These references indicate the appropriate appendix where code lists or code sources used as values for those elements may be found. Code lists and code sources are listed in data element number sequence in Appendixes A and B.

3.2 DATA ELEMENT SPECIFICATIONS
(Exerpted from ASC X12.3)

3 FREE-FORM MESSAGE
(SPEC: TYPE= AN MIN= 1: MAX= 60)
FREE-FORM TEXT.

19 CITY NAME
(SPEC: TYPE= AN MIN= 2: MAX= 19)
FREE-FORM TEXT FOR CITY NAME.

26 COUNTRY CODE
(SPEC: TYPE= ID MIN= 2: MAX= 2)
TWO CHARACTER IS0 STANDARD COUNTRY CODE
(SEE APPENDIX A.)

93 NAME
(SPEC: TYPE= AN MIN= 1: MAX= 35)
FREE-FORM ORGANIZATION NAME, OFFICIAL TITLE OR RELATED INFORMATION.

96 NUMBER OF INCLUDED SEGMENTS
(SPEC: TYPE= NO MIN= 1: MAX= 6)
TOTAL NUMBER OF SEGMENTS INCLUDED IN A TRANSACTION SET INCLUDING ST AND SE SEGMENTS.

116 POSTAL CODE
(SPEC: TYPE= ID MIN= 5: MAX= 9)
INTERNATIONALLY USED POSTAL ZONE CODE EXCLUDING PUNCTUATION AND BLANKS (ZIP CODE FOR UNITED STATES).

143 TRANSACTION SET IDENTIFIER
(SPEC: TYPE= ID MIN= 3: MAX= 3) UNIQUE IDENTIFYING NUMBER FOR THE TRANSACTION SET

156 STATE OR PROVINCE CODE
(SPEC: TYPE= ID MIN= 2: MAX= 2) STANDARD STATE/PROVINCE CODE DEFINED BY APPROPRIATE GOVERNMENTAL AGENCIES.

166 ADDRESS
(SPEC: TYPE= AN MIN= 1: MAX= 35)
ADDRESS INFORMATION

329 TRANSACTION SET CONTROL NUMBER
(SPEC: TYPE= AN MIN= 4: MAX= 9)
IDENTIFYING CONTROL NUMBER ASSIGNED BY THE ORIGINATOR FOR A TRANSACTION SET,

337 UTC TIME FILED
(SPEC: TYPE= TM MIN= 4: MAX= 4)
UNIVERSAL TIME OF THE SENDER OF THE TRANSMISSION SET EXPRESSED IN 24-HOUR CLOCK TIME (HHMM) (TIME RANGE: 0000 THROUGH 2359)

364 COMMUNICATION NUMBER
(SPEC: TYPE= AN MIN= 7: MAX= 21) COMPLETE COMMUNICATIONS NUMBER INCLUDING COUNTRY OR AREA CODE WHEN APPLICABLE.

373 DATE
(SPEC: TYPE= DT MIN= 6: MAX= 6)
DATE (YYMMDD)

3.3 DATA ELEMENT SPECIFICATIONS
(Proposed additions to ASC X12.3)

Q1 PRECEDENCE
(SPEC: TYPE= ID MIN= 1: MAX= 9)
PRECEDENCE (R, W, P OR EMERGENCY)

Q2 DESTINATION STATION OR POSTAL CODE
(SPEC: TYPE= ID MIN= 4: MAX= 10)
IDENTIFIER OF STATION MESSAGE IS TO BE DELIVERED TO.

Q3 MESSAGE NUMBER
(SPEC: TYPE= NO MIN= 1: MAX= 4)
NUMBER (BEGIN WITH 1 EACH MONTH OR YEAR)

Q4 HANDLING INSTRUCTION
(SPEC: TYPE= ID MIN= 3: MAX= 24)
HANDLING INSTRUCTIONS:
HXA - (FOLLOWED BY NUMBER.) COLLECT LANDLINE DELIVERY AUTHORIZED BY ADDRESSEE WITHIN . . . MILES. (IF NO NUMBER, AUTHORIZATION IS UNLIMITED.)
HXB - (FOLLOWED BY NUMBER.) CANCEL MESSAGE IF NOT DELIVERED WITHIN . . . HOURS OF FILING TIME; SERVICE ORIGINATING STATION.
HXC - REPORT DATE AND TIME OF DELIVERY (TOD) TO ORIGINATING STATION.
HXD - REPORT TO ORIGINATING STATION THE IDENTITY OF STATION FROM WHICH RECEIVED, PLUS DATE AND TIME. REPORT IDENTITY OF STATION TO WHICH RELAYED, PLUS DATE AND TIME, OR IF DELIVERED REPORT DATE, TIME AND METHOD OF DELIVERY.
HXE - DELIVERING STATION GET REPLY FROM ADDRESSEE, ORIGINATE MESSAGE BACK.
HXF - (FOLLOWED BY NUMBER.) HOLD DELIVERY UNTIL . . . (DATE).
HXG - DELIVERY BY MAIL OR LANDLINE TOLL CALL NOT REQUIRED. IF TOLL OR OTHER EXPENSE INVOLVED, CANCEL MESSAGE AND SERVICE ORIGINATING STATION.

Q5 RADIO CALLSIGN
(SPEC: TYPE= AN MIN= 4 : MAX= 10)
AMATEUR RADIO CALLSIGN.

Q6 CHECK
(SPEC: TYPE= NO MIN= 1: MAX= 4)
NUMBER OF WORDS/GROUPS IN TEXT ONLY.

Q7 PLACE OF ORIGIN
(SPEC: TYPE= AN MIN= 2: MAX= 25)
NOT NECESSARILY LOCATION OF STATION OF ORIGIN.

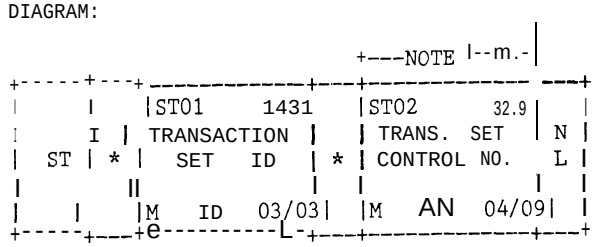
22.3.2 DATA SEGMENT DIAGRAMS
(Exerpted from ASC X12.22)

STANDARD REQUIREMENTS

ST TRANSACTION SET HEADER

PURPOSE: THE FIRST SEGMENT OF EACH TRANSACTION SET, CONTAINING THE TRANSACTION SET IDENTIFIER AND CONTROL NUMBER.

DIAGRAM:



NOTE: 1. THE "TRANSACTION SET CONTROL NUMBER" ENTRY IN THIS HEADER MUST MATCH THE '*TRANSACTION SET CONTROL NUMBER" ENTRY IN THE TRANSACTION SET TRAILER (SE).

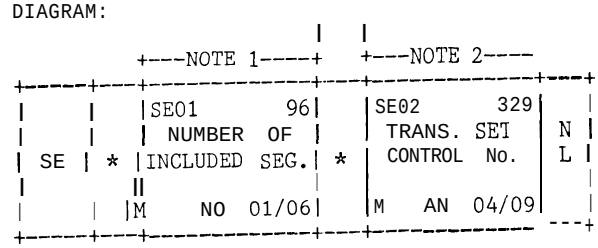
THE TRANSACTION SET IDENTIFIER (ST01) IS INTENDED FOR USE EY THE TRANSLATION ROUTINES OF THE INTERCHANGE PARTNERS TO SELECT THE APPROPRIATE TRANSACTION SET DEFINITION (e.g., QNU, SELECTS THE AMATEUR RADIO MESSAGE TRANSACTION SET).

STANDARD REQUIREMENTS

SE TRANSACTION SET TRAILER (END)

PURPOSE: THE LAST SEGMENT OF EACH TRANSACTION SET, CONTAINING THE NUMBER OF INCLUDED SEGMENTS AND TRANSACTION SET CONTROL NUMBER.

DIAGRAM:



NOTES: 1. THE "NUMBER OF INCLUDED SEGMENTS" IS THE TOTAL OF ALL SEGMENTS USED IN THE TRANSACTION SET INCLUDING THE (ST) AND (SE) SEGMENTS.

NOTES: 2. THE TRANSACTION SET CONTROL NUMBER VALUE IN THIS TRAILER MUST MATCH THE SAME ELEMENT VALUE IN THE TRANSACTION SET HEADER (ST).SEGME:NTS.

SE IMMEDIATELY FOLLOWS THE LAST SEGMENT OF EACH TRANSACTION SET.

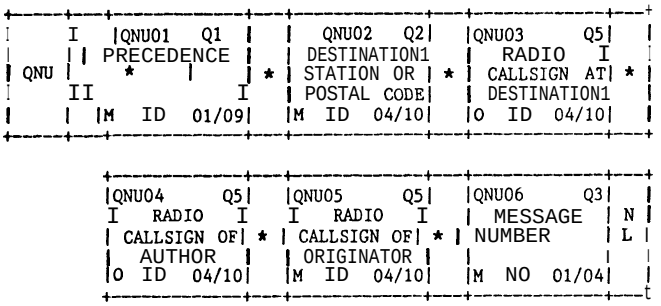
22.3.2 DATA SEGMENT DIAGRAMS
(Proposed additions to ASC X12.22)

STANDARD REQUIREMENTS

QNU AMATEUR RADIO PACKET MESSAGE HEADER

PURPOSE: THIS IS USED TO START AN AMATEUR RADIO MESSAGE TRANSACTION SET.

DIAGRAM:

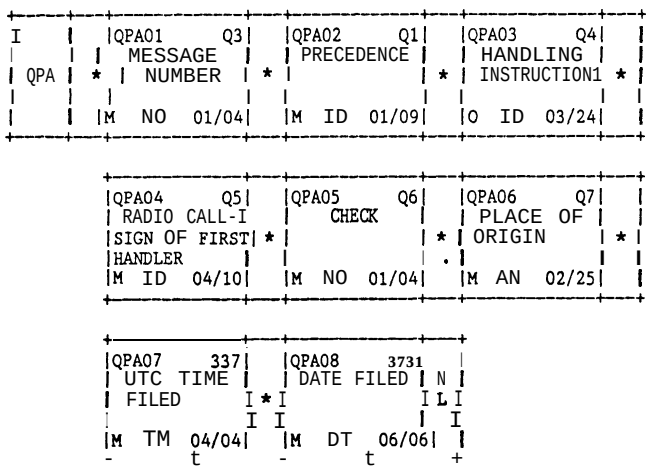


STANDARD REQUIREMENTS

QPA AMATEUR RADIO PACKET MESSAGE PREAMBLE

PURPOSE: THIS IS USED TO DEFINE AN AMATEUR RADIO MESSAGE PREAMBLE SECTION.

DIAGRAM:

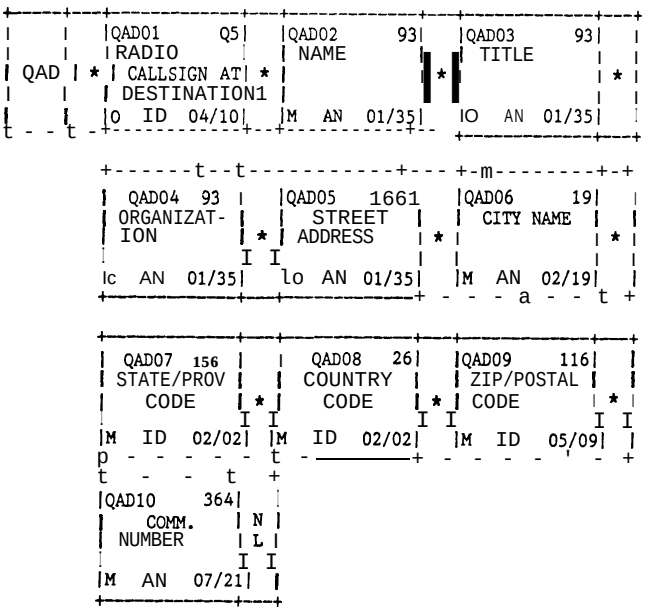


STANDARD REQUIREMENTS

QAD AMATEUR RADIO PACKET MESSAGE ADDRESS

PURPOSE: THIS IS USED TO DEFINE AN AMATEUR RADIO MESSAGE ADDRESS SECTION.

DIAGRAM:

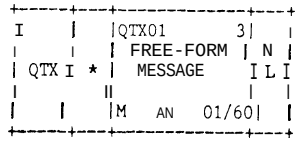


STANDARD REQUIREMENTS

QTX AMATEUR RADIO PACKET MESSAGE TEXT

PURPOSE: THIS IS USED TO DEFINE AN AMATEUR RADIO MESSAGE TEXT SECTION.

DIAGRAM:

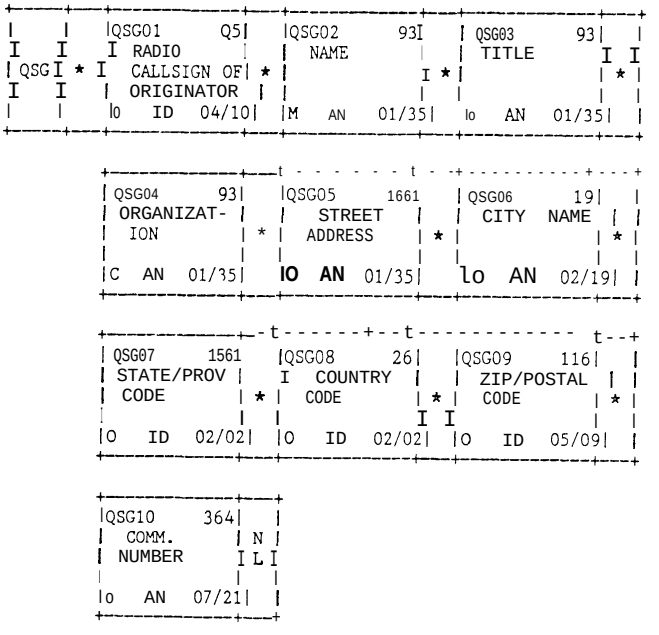


STANDARD REQUIREMENTS

QSG AMATEUR RADIO PACKET MESSAGE SIGNATURE

PURPOSE: THIS IS USED TO DEFINE AN AMATEUR RADIO MESSAGE SIGNATURE SECTION.

DIAGRAM:

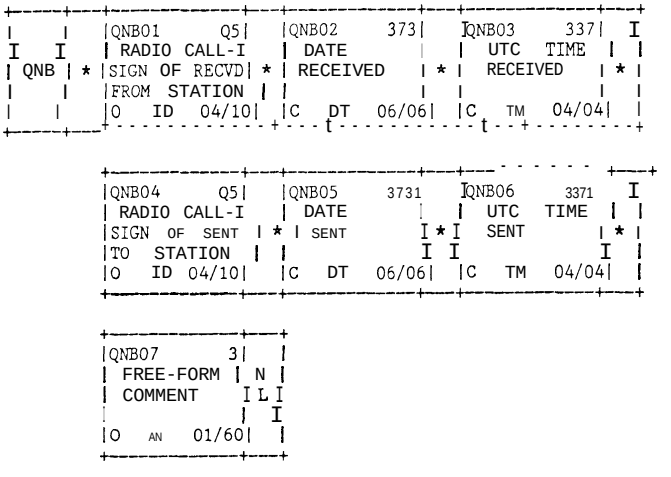


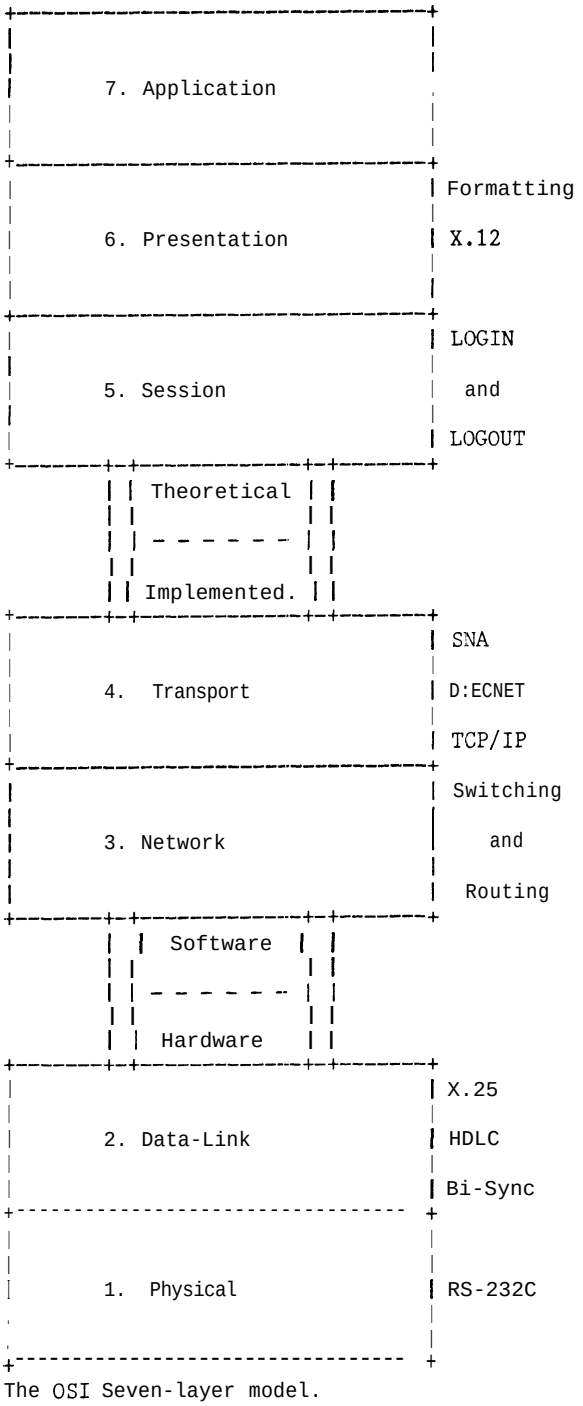
STANDARD REQUIREMENTS

QNB AMATEUR RADIO PACKET MESSAGE RELAY IDENTIFICATION

PURPOSE: THIS IS USED TO DEFINE AN AMATEUR RADIO MESSAGE RELAY OF MESSAGE DOCUMENTATION.

DIAGRAM:





The OSI Seven-layer model.

The International Standards Organization% Open System Interconnection (OSI) model divides local area network architecture into seven layers. Each layer in the model is defined and provides rules for network design. Viewed another way, the bottom four layers define the network and how it functions. The top three layers define how the network is used.

A M A T E U R M E S S A G E F O R M

THE AMERICAN RADIO RELAY LEAGUE
R A D I O G R A M
via amateur radio

NUMBER	PRECEDENCE	HX	STATION OF ORIGIN	CHECK	PLACE OF ORIGIN	TIME FILED	
1	R	B24	W1AW	8	NEWINGTON CONN	18302	Jul1

TO
DONALD SMITH
164 EAST SIXTH AVE
NORTH RIVER CITY MO 00789
733 4968

HAPPY BIRTHDAY X SEE YOU SOON X LOVE

DIANA

A M A T E U R P A C K E T M E S S A G E

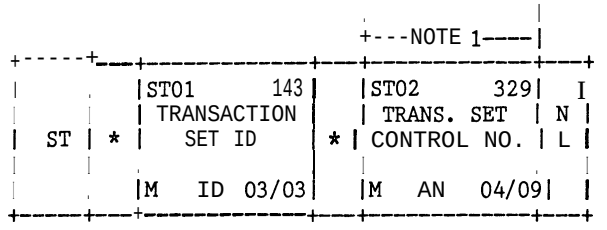
```
ST*QNU*0008
--------------------------------------------------------------------------------
 QNU*R*00789***W1AW*1
QPA*1*R*HXB24*W1AW*8*NEWINGTON CONN*1830*850701
 QAD**DONALD SMITH***1645 EAST SIXTH AVE*NORTH R
   IVER CITY*MO*US*00789*7334968
 QTX*HAPPY BIRTHDAY X SEE YOU SOON X LOVE
 QSG**DIANA********
 QNB*ORIGINATE*860808*1441*NC4E*861217*2230*ORIGINATED by W1AW Newington, CT
 --------------------------------------------------------------------------------
 SE*8*0008
```

Cheat Sheet for ARRL Message format using the International Xl2 Protocall

```
        TRANSACTION SET HEADER
ST*QNU* +
Trans.  set  control  no.   (a  unique  identifying  number)
```

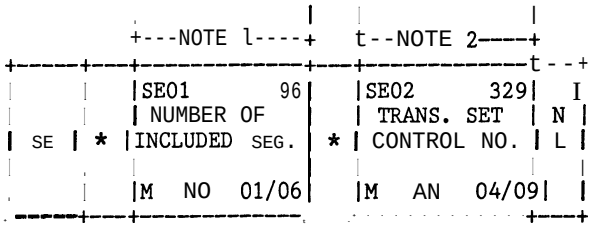
```
        AMATEUR RADIO PACKET MESSAGE HEADER
QNU* +
Precedence + * +
Destination station or postal code + * +
Radio callsign at destination + * +
Radio callsign of author + * +
Radio callsign of originator + *
Message number

        AMATEUR RADIO PACKET MESSAGE PREAMBLE
QPA* +
Message number + * +
Precedence + * +
Handling Instruction + * +
Radio callsign of first handler + * +
Check + * +
Place of Origin + * +
UTC time filed + * +
Date Filed

        AMATEUR RADIO PACKET MESSAGE ADDRESS
QAD* +
Radio callsign at destination + * +
Name + * +
Title + * +
Organization + * +
Street address + * +
City name + * +
State/Prov code + * +
Country code + * +
Zip/Postal Code + * +
Comm number

        AMATEUR RADIO PACKET MESSAGE TEXT (UP TO 99 RECORDS MAXIMUM)
QTX* +
Free-form message

        AMATEUR RADIO PACKET MESSAGE SIGNATURE
QSG* +
Radio callsign of originator + * +
Name + * +
Title + * +
Organization + * +
Street address + * +
City name + * +
State/Prov code + * +
Country code + * +
Zip/Postal Code + * +
Comm number
```

```
        TRANSACTION SET TRAILER (END)
SE* +
Number of included seg + * + (Tot of all records in message, including ST & SE)
Trans. set control no.        (same identifying number as in ST* record)
```



NOTE: 1. THE "TRANSACTION SET CONTROL NUMBER" ENTRY IN THIS HEADER MUST MATCH THE "TRANSACTION SET CONTROL NUMBER" ENTRY IN THE TRANSACTION SET TRAILER (SE).

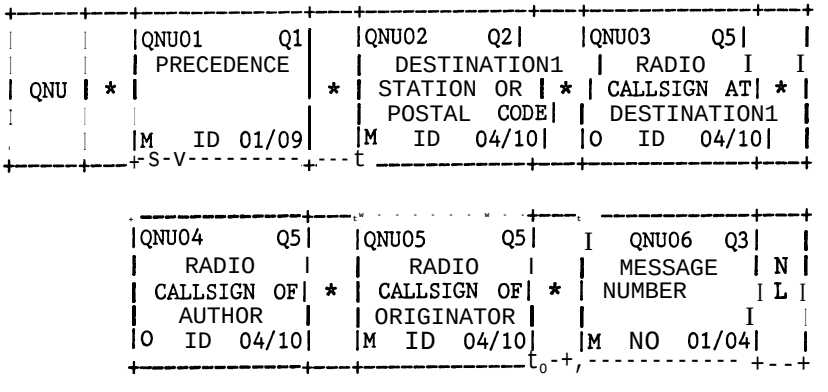
THE TRANSACTION SET IDENTIFIER (STO1) IS INTENDED FOR USE BY THE TRANSLATION ROUTINES OF THE INTERCHANGE PARTNERS TO SELECT THE APPROPRIATE TRANSACTION SET DEFINITION (e.g., NUO, SELECTS THE AMATEUR RADIO MESSAGE TRANSACTION SET).

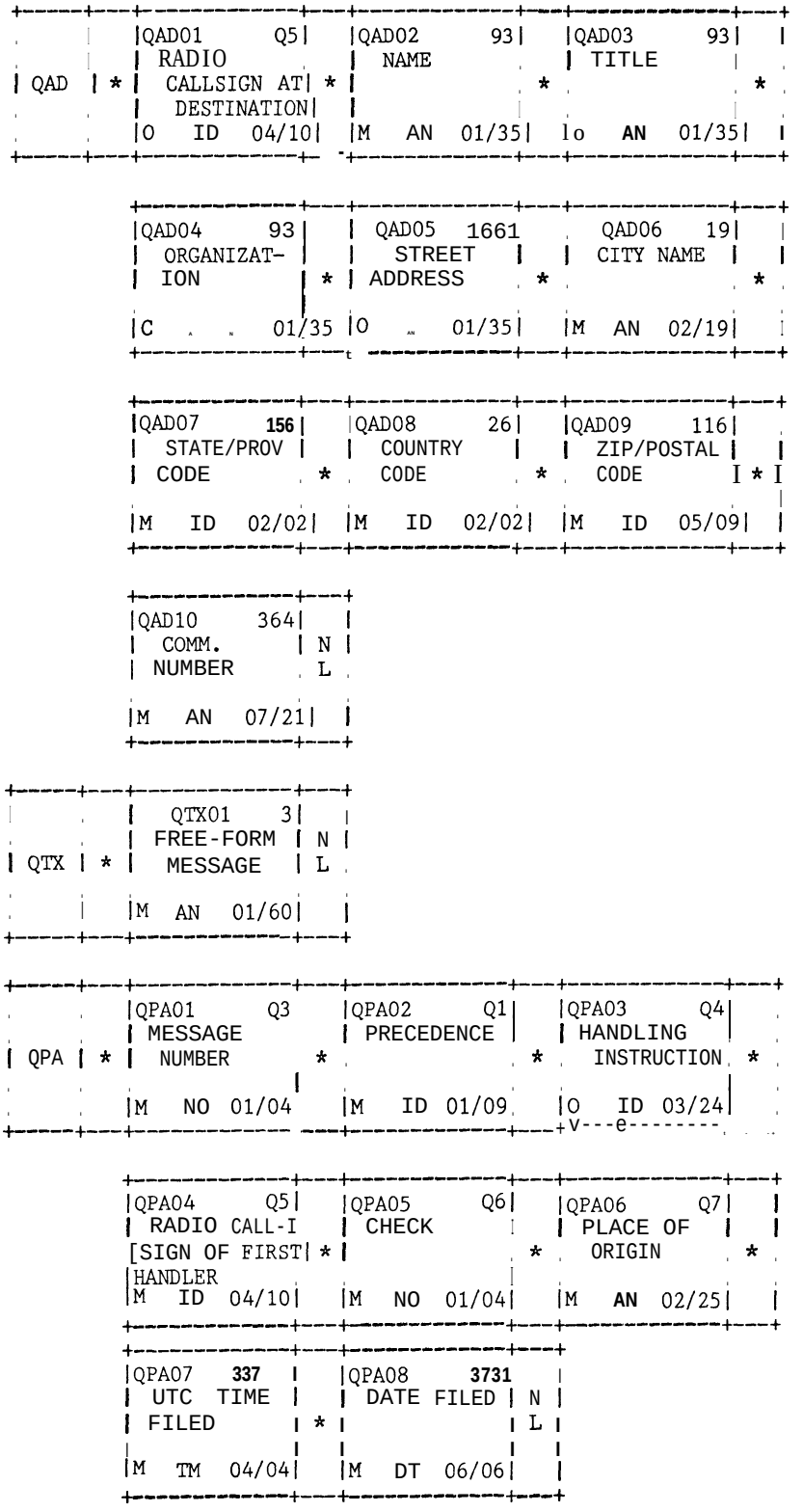


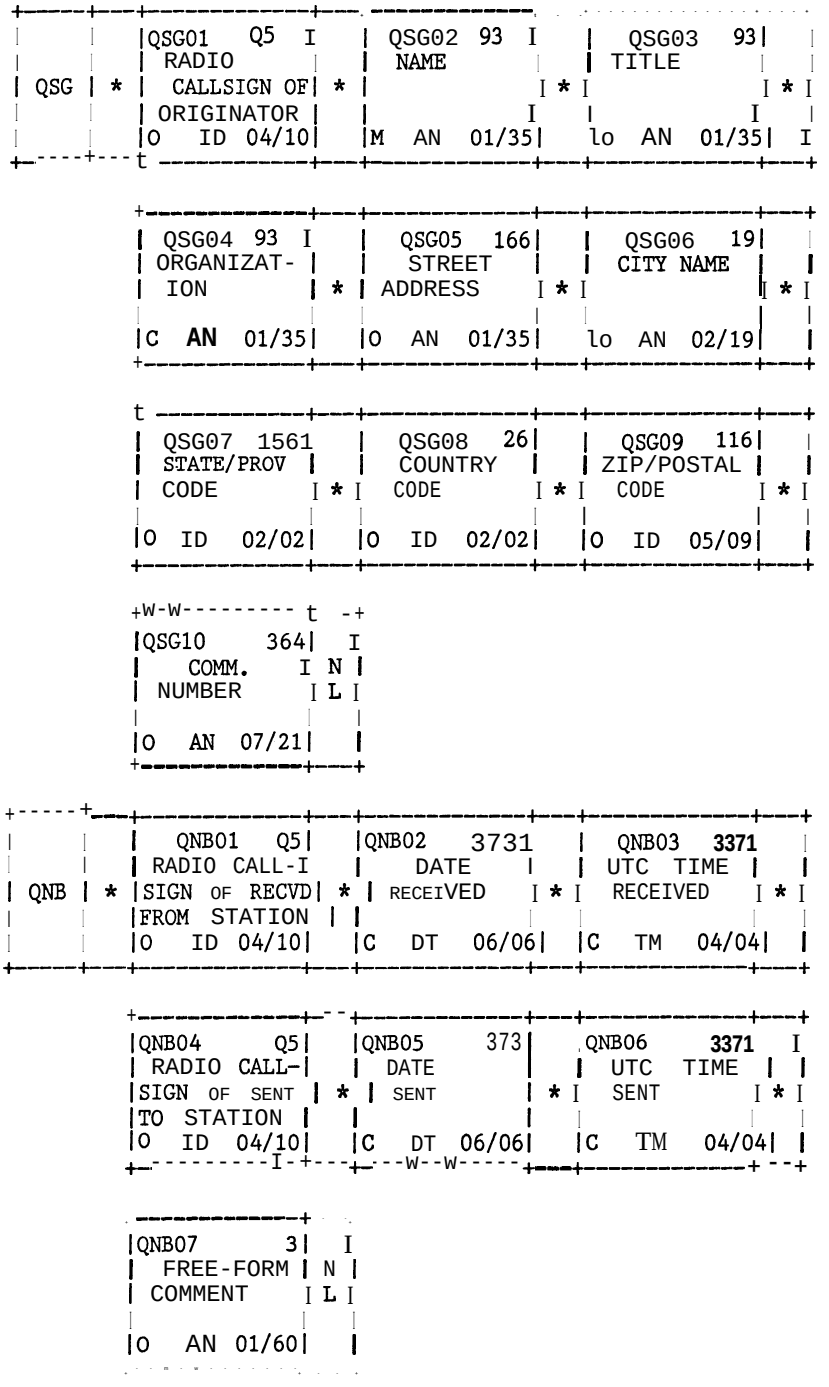
NOTES: 1. THE "NUMBER OF INCLUDED SEGMENTS" IS THE TOTAL OF ALL SEGMENTS USED IN THE TRANSACTION SET INCLUDING THE (ST) AND (SE) SEGMENTS.

NOTES: 2. THE TRANSACTION SET CONTROL NUMBER VALUE IN THIS TRAILER MUST MATCH THE SAME ELEMENT VALUE IN THE TRANSACTION SET HEADER (ST).SEGMENTS.

SE IMMEDIATELY FOLLOWS THE LAST SEGMENT OF EACH TRANSACTION SET.




QAD
QAD01 Q5
RADIO CALLSIGN AT DESTINATION
O ID 04/10
QAD02 93
NAME
M AN 01/35
QAD03 93
TITLE
O AN 01/35
QAD04 93
ORGANIZAT- ION
C 01/35
QAD05 1661
STREET ADDRESS
O 01/35
QAD06 19
CITY NAME
M AN 02/19
QAD07 156
STATE/PROV CODE
M ID 02/02
QAD08 26
COUNTRY CODE
M ID 02/02
QAD09 116
ZIP/POSTAL CODE
M ID 05/09
QAD10 364
COMM. NUMBER
N L
M AN 07/21
QTX
QTX01 3
FREE-FORM MESSAGE
N L
M AN 01/60
QPA
QPA01 Q3
MESSAGE NUMBER
M NO 01/04
QPA02 Q1
PRECEDENCE
M ID 01/09
QPA03 Q4
HANDLING INSTRUCTION
O ID 03/24
QPA04 Q5
RADIO CALL-I [SIGN OF FIRST HANDLER
M ID 04/10
QPA05 Q6
CHECK
M NO 01/04
QPA06 Q7
PLACE OF ORIGIN
M AN 02/25
QPA07 337
UTC TIME FILED
M TM 04/04
QPA08 3731
DATE FILED
N L
M DT 06/06


QSG
QSG01 Q5 RADIO CALLSIGN OF ORIGINATOR
O ID 04/10
QSG02 93 NAME
M AN 01/35
QSG03 93 TITLE
lo AN 01/35
QSG04 93 ORGANIZATION
C AN 01/35
QSG05 166 STREET ADDRESS
O AN 01/35
QSG06 19 CITY NAME
lo AN 02/19
QSG07 1561 STATE/PROV CODE
O ID 02/02
QSG08 26 COUNTRY CODE
O ID 02/02
QSG09 116 ZIP/POSTAL CODE
O ID 05/09
QSG10 364 COMM. NUMBER
O AN 07/21
QNB
QNB01 Q5 RADIO CALL-SIGN OF RECVD FROM STATION
O ID 04/10
QNB02 3731 DATE RECEIVED
C DT 06/06
QNB03 3371 UTC TIME RECEIVED
C TM 04/04
QNB04 Q5 RADIO CALL-SIGN OF SENT TO STATION
O ID 04/10
QNB05 373 DATE SENT
C DT 06/06
QNB06 3371 UTC TIME SENT
C TM 04/04
QNB07 3 FREE-FORM COMMENT
O AN 01/60

DESIGN ABSTRACTIONS FOR PROTOCOL SOFTWARE

Paul Taylor, VK3BLY
P.O. Box 118
Eltham, 3095
Victoria
Australia

Abstract

As amateur packet radia software becomes more complicated and software development environments improve, the use of high level languages will become more favourable. A design approach for protocol software based on modules and Finite State Machines is described, which formalises the interface to IO devices, and extends the use of the protocol's State Machine model into the implementation stage. Its adoption should make implementations of Level 2 and 3 protocols quicker, easier and more understandable.

Introduction

Traditionally, communication software for microprocessors was written in the native assembly language, for two reasons; the simple microprocessor architectures limited the code complexity, size and speed, and the environment needed for software development at a higher level was not available. The early Terminal Node Controllers were implemented under these restrictions.

The first objection is no longer valid. The explosion of the personal computer has made significant processing power cheap and readily available, and many of these machines can be used for both software development and as dedicated or shared host.

Secondly, software tools and environments have grown into rich and Plentiful arena to develop software in. Compilers, interpreters, simulations, graphics, databases and communications software of surprising complexity and usefulness are now available for even the simplest of microprocessors.

Protocol software developers should consider adopting high level languages (such as Pascal and C) for their communications systems. The advantages of expressing software in a structured, high level language are well known (see any Software Engineering text).

Once a high level language has been chosen, some important design considerations must be faced. The features of the chosen language must be exploited to improve the software design, implementation and maintenance. Put more generally, the problem may be worded as a question; how should a communications program be designed? A design approach which answers this question will be discussed.

Data Abstraction

All software for driving devices (ie. ports) should be modularised. High level languages offer mechanisms for successive abstraction or modularisation, usually in the form of procedures/ functions or subroutines. A design approach enforces the decomposition of software into modules in a discipline4 way.

Modularisation allows for the implementation of well defined interfaces to entire functionally-independent components of a software system. A module (within a program) is one or more sub-programs which Perform a specified task on some data. The module encapsulates both the data object, and the sub-programs which manipulate it,. All aspects of the module are completely defined by the programmer, in an appropriate notation.

Definition of a Module

An example drawn from packet radio software is shown below. Modules are defined for a serial port (ie. a UART), and a "packet_port" (ie. the software interface to a protocol controller chip),

These definitions are presented in a Pascal-like PSeudoCode, a form which has been successfully used by the author. The purpose of the notation is to convey semantic specification, rather than syntactic detail.

```
serial_port:
 {global data3
 registers:
   begin
     TXbuff, RXbuff, DivisorLSB,
     DivisorMSB, Intrpt, LineControl,
     LineStat, ModemControl,
     ModemStat:UART_Register;
   end
 {operators  }
   procedure initialise;
   procedure send_char (ch:char) ;
   function  char_received:boolean;
   function  get_rece i ved,char :char ;
end {serial_port};

packet_port:
   {global data}
   buffers:array[1..n] o f  buffer_type;
   registers:
    begin
      TXbuff, RXbuff, Intrpt, LineControl,
      LineStat, ModemControl, ModemStat
      :Protocol_Controller_Register;
    end
   {operators  }
   procedure initi3lise;
   procedure send_frame(b:buffer);
   procedure listen(b:buffer);
   function frame_received:boolean;
   procedure get_received_frame(
                     var b:buffer) ;
end {packet_port};
```

Once the module (port interface) has been specified 3s the examples show, implementation of the module can Proceed by further refining the data objects, and by writing the interface procedures or function code. If necessary, the module specification can be modified slightly in the light of implementation considerations but this should be avoided since it implies weakness in the original module specification.

Once implemented, using the module should be a simple case of making calls to the interface procedures or functions 3s required. Since languages such as Pascal and C do not support modules explicitly, it is the programmer's responsibility to enforce correct use of the module in his or her code, by using the defined interface5 only. For instance, when using a Packet-port module, there must be no reading or writing of any of the data variables, or the port device itself.

The advantages and disadvantages of using this organisation of device interface software will be discussed later.

Control Abstraction

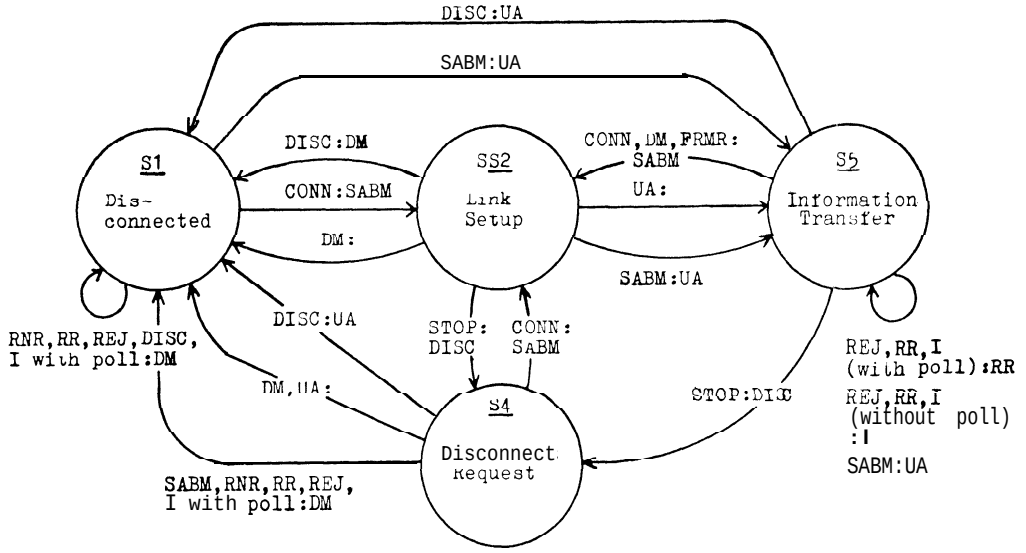
Communication systems are usually "event driven"; the software must respond to asynchronous events, such as an operator keyed command, or a received Packet. Such software must decide on appropriate processing of events on the basis of the history, or state of the system. A form3l model which is applicable in such systems is the Finite State Machine.

A Finite State Machine (FSM) consists of a finite set of states, each of which knows of a set of events, and 3 transition function which maps (state, event) into (new-state, action), The action is a member of 3 set of actions, which express interaction or modification of the FSM's environment.

FSMs are easily depicted in diagrammatic form. The states are represented by circles, transitions between states by directed arcs, and events 3nd action5 by "event:action" labels on the transitions. Part of the FSM model of AX25 [Fox 843 is presented in Figure 1. It shows the states which are passed through when the local station connects to a remote station.

The following steps take place when a connection is established. Initially, our station is in the "Disconnected" state. Our attempt to connect to another station (by typing "CONN <callsign>") is interpreted by the FSM as 3n event, 3nd classified (e19) The FSM state transition function is invoked, with Parameters current-state (which is "Disconnected"), and event (which is e19). The function yields a new state ("Link Establishment"), and an action (send 3 SABM packet to the addressed station). The action is Performed and the system waits, ready to respond to my of the events which may occur in the "Link. Establishment" state.

From this simplified example, it should be clear that 3n FSM model is ideal for specification, de5ign and implementation of 3n event driven system, such 3s a Protocol. The advantages of adoption of the FSM model are discussed later. The rest of this section describes some ideas on how FSMs can be implemented in Pascal, arepresentative high level language.



Note: CONN and STOP are operator commands for connecting and disconnecting to the remote station respectively.
All other names are AX.25 frame types.

AX.25 FSM EVENT / STATE	e0 I with Poll	e8 SABM either	e9 DISC either	I e16 UA either	e17 DM either	e19 CONN cmd
1 Disconnected	DM	UA,S5	DM	DM	DM	SABM,S2
2 Link. Setup	-	UA,S5	DM,S1	-	-	-
3 Frame Reject	FRMR	UA,S5	UA,S1	-		SABM,S2
4 Disconnect Rqst	DM,S1	DM,S1	UA,S1	S1	S1	SABM,S2
5 Information Xfer	R R	UA	UA,S1	-	-	SABM,S2
. .						

Figure 1. Extract of the AX.25 FSM.

FSM based Implementation of AX.25

Once the FSM model of the protocol is completely defined, implementation becomes straightforward and somewhat, mechanical. The components to be defined are:

i. the state table,

ii. the state variable,

iii. the action procedures,

iv. the FSM procedure,

v. the mainloop.

Some examples of these follow. The Finite State Table (FST) can easily be defined:

```
const
   S1_Disconnected=1;
   S2_LinkEstablishment=2;
      .
   A0_NoAction=0;
   A1_SendSABM=1;
      .
   E1_CONNcmd=1;
   E2_DMreceived=2;
      .
   nbr_states =n;
   nbr_events =n ;
   nbr_actions=n;

type
   state_type  =
         S1_Disconnected..Sn_StateN;
   event-type  =
         E1_CONNcmd..En_EventN;
   action-type =
         A0_NoAction..An_ActionN;
```

```
var
   FST:array[1..nbr_states,
             1..nbr_events] of
             record
                newstate:state_type;
                action  :action-type
             end;
```

The state variable is simply defined:

```
var  current_state:state_type;
```

Each (newstate, event) pair in the FST must be initialised. In Pascal, this becomes rather tedious:

```
FST[S1_Disconnected,
    E1_CONNcmd].newstate:=
            S2_LinkEstablishment;
FST[S1_Disconnected,
    E1_CONNcmd].action  :=
            A1_sendSABM;

FST[S2_LinkEstablishment,
    E2_DMreceived].newstate:=
            S1_Disconnected;
FST[S2_LinkEstablishment,
    E1_DMreceived].action  :=
            A0_NoAction;

      etc
       .
```

The state transition function can be implemented as a simple Pascal function:

```
function FSM(event:event_type):state_type;
{ globals: FST, current_state }
begin
  case FST[current_state, event], action of
  A0_NoAction: { do nathing }
  A1_sendSABM: begin sendSABM(,,..,,) end;
  A2_Action2 : begin { do Action 2 } end;
       .
  An_ActionN : begin { do Action n } end;
  end {case};
  FSM :=
    FST[current_state, event].newstate;
end (FSM);
```

The action procedures A0_NoAction ... An_ActionN take care of all Processing needed to handle the event. These Procedures may call other 'utility' procedures to do lower level processing.

The main loop of the Process takes the following form:

```
current-state := initial-state;
repeat
   get-event (event);
   current_state := FSM(event);
until current-state = terminal_state;
```

The procedure "get_event (event)" monitors both the packet_port, and the keyboard, for any input which can be classified as an event:

```
procedure get_event (var event:event_type);
var
   kbd_buffer:string[1..longest_cmd];
   frame_buffer:frame;

event := no-event;
repeat
  {wait for received frame or typed key}
  if Key Pressed then
     get char
     if char is CR then
        analyse_cmd(kbd_buffer,event)
     else
        append char to  kbd_buffer;
  else
     if frame_received then
        analyse_frame(frame_buffer,event)
until event <> no_event;
```

The comment "wait on a received frame or typed key" refers to an implementation specific issue. If both frame reception and keyboard monitaring is done by interrupt handlers, Procedure get-event should unschedule itself at this Point. In an implementation where interrupts are not available, the procedure can simply loop until 3 received frame is detected or a key is pressed.

Experience with Design Abstractions

Experience has shown that adoption of these design abstractions has many more advantages than disadvantages.

Object-based implementations of device interface software (such as the serial_port or packet_port) offer all the advantages of modularity:

i. Abstraction. Specific details of the physical port device are hidden.

ii. A well defined interface. All references to the device go through a common interface.

iii. Interface definitions can be application specific. The definition of the interface to the Port (ie, the access Procedures) is entirely up to the programmer.

iv, Isolation. Replacement or upgrading of the physical device can be done transparently to the rest of the software.

A possible disadvantage is the overhead of structure. Using structure imposes the need for more object code and data, which may in some cases result in slower execution speed. This has not been a problem in both RTTY and AX.25 software, but may cause problems at higher baudrates.

The advantages of adopting the FSM model in implementation were found to be:

i. Centralisation of Control. The entire 'control policy' of the protocol is described neatly and simply in a single table.

ii. Simplicity. A complicated Protocol can become readable and understandable.

iii. Enforcement of Structure. Use of an FSM model enforces a uniform approach to the organisation of the entire program.

iv. Modifiability. Maintenance of a Piece of software is made easier by the strict enforcement of structure. Once the Finite State Table has been set up, and the set of action procedures written as drivers for lower level "utility procedures", new actions, transitions and states can usually be added quickly and easily.

The only disadvantage encountered was the Potential for the F'SM model to become complex. When protocols become complicated, their FSM model can become very large and cumbersome (this is called "state explosion"). Tabular representations of large FSMs can be managed, but graphical representations may be unwieldly.

Conclusion

The average personal computer can host a perfectly satisfactory environment for communication software development in a high level language.

The use of a high level language requires a more formal approach to software design. Software for driving the IO devices can conveniently be organised as a module. In an event driven system, the entire system's control flow can be encapsulated in a Finite State Machine model,

These design abstractions have been applied in the development of software for RTTY and AX.25 by the author. They were found to increase software reliability, and make the source code more understandable and modifiable.

References

[Fox 84] Terry L. Fox (WB4JFI), "AX.25 Amateur Packet-Radio Link-Layer Protocol", October 1984.